Carreira Dev

Hábitos para evoluir no desenvolvimento de software para além do código

Casa do Código | alura

JÉSSICA NATHANY

Sumário

- ISBN
- Prefácio
- Sobre a autora
- Agradecimentos
- Parte I: Entre códigos, bugs e cafés – O trabalho por trás da construção de software
  - 1 Colaboração e papéis: conhecendo as responsabilidades no time de desenvolvimento de software
  - 2 Desenvolvedores não só escrevem códigos
  - 3 Metodologias Ágeis
  - 4 De Junior a Sênior: erros, acertos e a paciência para escalada do sucesso
- Parte II: 10 hábitos que te tornarão um(a) desenvolvedor(a) eficiente
  - 5 O hábito
  - 6 Hábito 1: Crie projetos desafiadores e construa um excelente portfólio
  - 7 Hábito 2: Aprimore sua comunicação
  - 8 Hábito 3: Organize seus estudos e foque no essencial
  - 9 Hábito 4: Domine o inglês, a língua da tecnologia
  - 10 Hábito 5: Participe e contribua com a comunidade Tech
  - 11 Hábito 6: Primeiro entenda os conceitos, depois consolide com a prática
  - 12 Hábito 7: Alinhe suas habilidades técnicas ao entendimento do negócio
  - 13 Hábito 8: Lições que você pode aprender e ensinar durante o code review
- Parte III: O mercado e o futuro da carreira Dev
  - 14 Tendências e investimentos em IA no mercado tech

- 15 Ferramentas de IA que potencializam o trabalho dos devs
- 16 Como se preparar para entrevistas técnicas e comportamentais
- 17 Equilíbrio sustentável: Como prosperar na carreira tech sem perder a si mesmo
- 18 Referências bibliográficas

ISBN

Impresso: 978-85-5519-403-0

Digital: 978-85-5519-404-7

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

Nathany, Jéssica
Carreira dev : hábitos para evoluir no desenvolvimento de software para além do código / Jéssica Nathany. -- 1. ed. -- São Paulo : AOVS Sistemas de Informática, 2025.

Bibliografia.
ISBN 978-85-5519-403-0

1. Ciência da computação 2. Engenharia de software 3. Programação (Computadores) 4. Software aplicativo - Desenvolvimento I. Título.

25-286583 CDD-004.2

Índices para catálogo sistemático:

1. Software : Arquitetura, projeto e desenvolvimento : especificações técnicas : Ciência da computação 004.2

Aline Graziele Benitez - Bibliotecária - CRB-1/3129

Caso você deseje submeter alguma errata ou sugestão, acesse http://erratas.casadocodigo.com.br.

Prefácio

Nunca a Engenharia de Software foi tão essencial quanto agora. Mas o que realmente significa ser um(a) desenvolvedor(a) de software na atualidade? Por que existe essa necessidade constante de aprender e evoluir? A comunidade de desenvolvedores, muitas vezes referida como "bolha dev", pode apresentar discordâncias e conflitos de ego; contudo, sua unidade e solidariedade são sem precedentes.

Vivemos em um mundo acelerado, marcado por mudanças tecnológicas contínuas. O papel das pessoas desenvolvedoras vai muito além da escrita de códigos. Ser desenvolvedor(a) hoje exige pensamento analítico, comunicação eficaz, domínio de novas ferramentas tecnológicas e do negócio, além de uma conexão real com a comunidade para colaborar, inovar e causar impacto no mundo. Como navegar por tudo isso de forma proveitosa e saudável?

Para quem é este livro?

Este livro é destinado tanto a iniciantes que estão começando sua jornada na programação quanto às pessoas desenvolvedoras que desejam aprimorar suas habilidades interpessoais e expandir sua visão de carreira. O pré-requisito para a leitura é ter uma base mínima de programação, curiosidade e vontade de crescer profissionalmente.

Se você está no início, encontrará aqui um guia prático para construir uma base sólida. Se já tem experiência, as reflexões e provocações ao longo dos capítulos podem inspirar você a enxergar a profissão sob novas perspectivas.

Como este livro está estruturado

O livro está dividido em três partes principais, cada uma explorando aspectos fundamentais da carreira em desenvolvimento de software:

Parte 1 – Entre códigos, bugs e cafés – O trabalho por trás da construção de software: Começamos explorando os pilares essenciais da jornada dev, como metodologias ágeis, a importância da paciência no processo de aprendizado e a construção de uma base consistente.

Parte 2 – Hábitos que te tornarão um(a) desenvolvedor(a) eficiente: Conheceremos hábitos fundamentais para crescer de forma estruturada na profissão. Vamos abordar boas práticas de programação, a importância de aprender um novo idioma, aprendizado contínuo, comunidade, comunicação e muito mais.

Parte 3 – O mercado e o futuro da carreira dev: Entramos no mercado de trabalho e exploramos as tendências tecnológicas com o uso de inteligência artificial (IA). Veremos desde a preparação para entrevistas técnicas, tipos de entrevistas para programadores, até equilíbrio entre vida pessoal e profissional. São ideias para navegar melhor pelo mercado e construir uma carreira sustentável.

Uma jornada além do código

Mais do que ensinar técnicas, o livro convida você a refletir sobre o papel humano por trás da construção do software. Compartilho aqui aprendizados da minha trajetória e trago depoimentos e experiências de outros profissionais que ajudaram a enriquecer o conteúdo deste livro com pontos de vista diferentes.

Vamos além das competências técnicas e abordamos habilidades interpessoais, a importância da colaboração, da comunidade e da comunicação, que são elementos tão fundamentais quanto saber programar.

Espero que este livro acompanhe você em sua caminhada, trazendo ideias práticas, provocações e reflexões que farão diferença no seu crescimento profissional e também como pessoa.

Sobre a autora

Meu nome é Jéssica Nathany. Sou formada em Sistemas de Informação pela Faculdade Campos Salles (2014) e pós-graduada em Arquitetura e Desenvolvimento de Software pela FIAP (2017). Tenho 14 anos de experiência na área de Tecnologia da Informação e 12 anos dedicados ao desenvolvimento de sistemas.

Ao longo da minha trajetória, atuei com diferentes tipos de projetos — desde sistemas CRM, e-commerce e banco digital até plataformas de entretenimento voltadas à gravação de novelas. Atualmente, trabalho em um produto bem específico, voltado para soluções empresariais.

Sou movida por uma busca constante por evolução. Não me considero uma desenvolvedora "fora da curva" — pelo contrário, estudo intensamente para aprimorar minhas habilidades e acompanhar as transformações do setor. Tenho um grande apreço por fundamentos e conceitos técnicos, e gosto de aplicar esses conhecimentos em projetos práticos. Conversar com amigos da área sobre desafios, aprendizados, desabafos e até mesmo questões filosóficas da profissão é algo que me inspira e me traz ideias.

Além de atuar como desenvolvedora de software, também produzo conteúdo para o podcast *Café Debug*, que apresento junto com o Wesley Fratini. O programa é dedicado à discussão de temas sobre ciência, tecnologia e desenvolvimento de software. Nele, participo da criação das pautas, da escolha dos convidados, da curadoria dos assuntos e do desenvolvimento do site oficial. Tudo isso com o objetivo de compartilhar conhecimento de forma acessível e com um toque de profissionalismo. Em resumo: respiro tecnologia 24 horas por dia.

O podcast me proporcionou diversas conexões com pessoas da área e o envolvimento com a comunidade foi o que me motivou a escrever este livro. Gosto de observar o que acontece ao meu redor, trocar ideias com profissionais da área, seja sobre temas técnicos ou

interesses pessoais, e refletir sobre como podemos evoluir juntos como desenvolvedores e como pessoas.

Aqui compartilho não apenas o que aprendi ao longo de mais de uma década de carreira, mas também as lições mais sutis: aquelas sobre crescimento, escolhas e a cultura vibrante da nossa indústria em constante transformação. Acredito no poder das ideias, da inovação e no impacto positivo que a tecnologia pode ter no mundo. Sigo aprendendo todos os dias. E como disse Cora Coralina: *"Feliz é aquele que transfere o que sabe e aprende o que ensina."*

Agradecimentos

Quero expressar minha profunda gratidão aos meus pais, que, mesmo com poucos recursos, sempre me incentivaram ao máximo nos estudos e me proporcionaram uma educação sólida.

Sou grata, também, a mim mesma por não ter desistido de perseguir meus objetivos, mesmo diante das adversidades e limitações enfrentadas ao longo da jornada. Mantive a constância, a disciplina e a perseverança acreditando que seria possível conquistar resultados significativos.

Agradeço, ainda, a todos que contribuíram para a realização deste livro — enriquecendo-o com experiências, opiniões valiosas e o generoso compartilhamento de seus aprendizados.

Parte I: Entre códigos, bugs e cafés – O trabalho por trás da construção de software

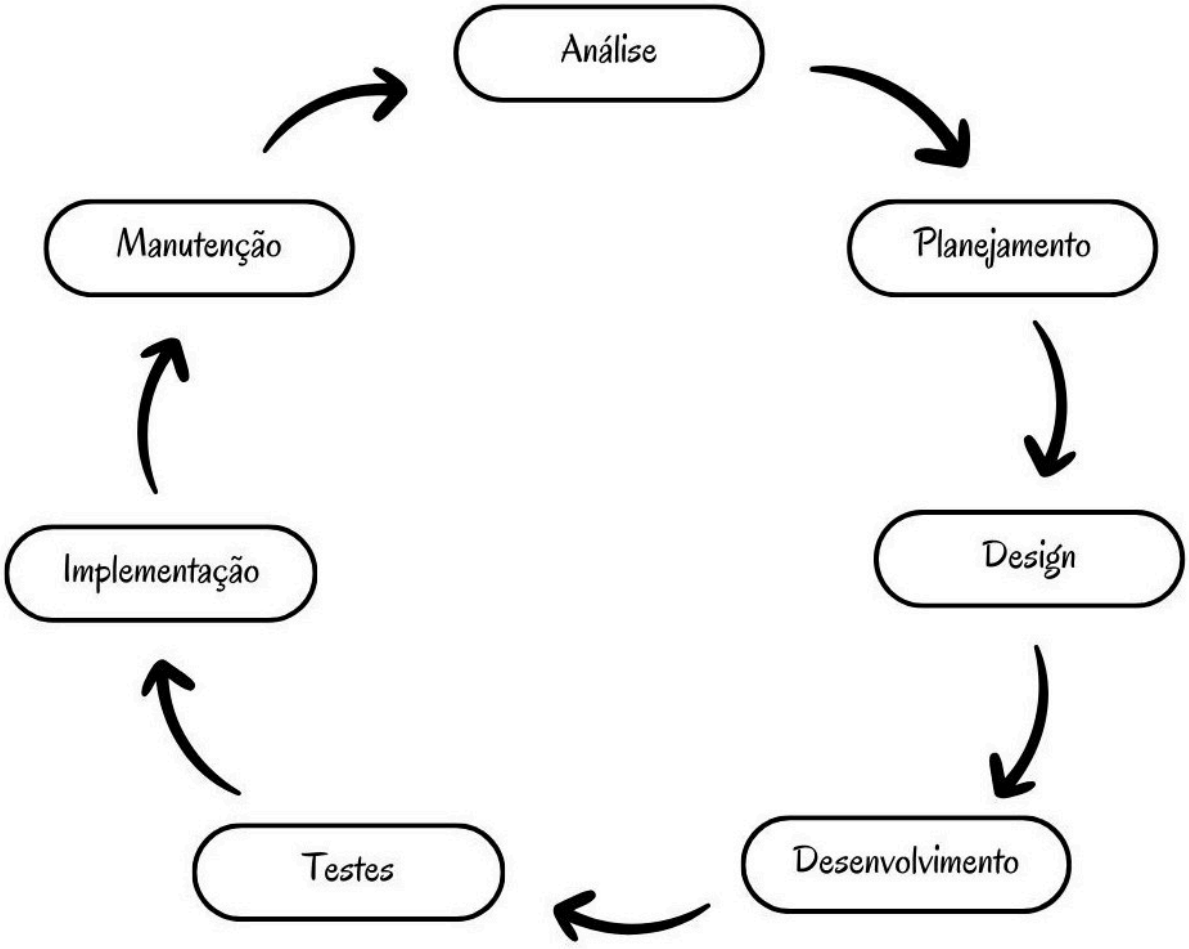

Figura 1: Ciclo de desenvolvimento de software.

Capítulo 1
Colaboração e papéis: conhecendo as responsabilidades no time de desenvolvimento de software

Figura 1.1: Colaboração e papéis: conhecendo as responsabilidades no time de desenvolvimento de software.

Vamos mergulhar no fascinante mundo do desenvolvimento de software, explorando as nuances e desafios que definem esta arte. Como pessoas desenvolvedoras, nossa jornada é repleta de códigos, bugs e, claro, muitos cafés. Mas, além dessa superfície agitada, o que mais se esconde no processo de criar software?

Gosto de usar a seguinte analogia: a criação de software é como a gastronomia. Imagine um restaurante onde os pratos são entregues aos clientes, assim como os softwares finalizados são disponibilizados para os usuários. Cada ingrediente escolhido pelo chef reflete uma decisão cuidadosa do(a) desenvolvedor(a) ao selecionar a linguagem que será usada e as ferramentas certas para o trabalho. A cozinha, por sua vez, representa o fluxo de trabalho organizado e colaborativo, no qual a comunicação é fundamental para a entrega bem-sucedida dos pratos em tempo hábil, assim como ocorre na criação do software. Para cada prato preparado, há ferramentas específicas a serem usadas: a faca para cortar peixe é diferente da faca para cortar carne ou legumes, assim como panelas e colheres variam de acordo com a função. Da mesma forma, na

construção do software, cada linguagem de programação tem suas peculiaridades e é adequada para situações específicas.

Confira alguns exemplos:

- Chef de cozinha / Tech lead: Ambos coordenam equipes e garantem que todos estejam cientes de suas responsabilidades, tarefas e papéis.
- Receita / Planejamento do projeto: Tanto na culinária quanto no desenvolvimento de software, é necessário um bom planejamento detalhado para assegurar o resultado desejado.
- Ingredientes de qualidade / Código limpo e eficiente: A escolha de ingredientes de alta qualidade reflete a importância de um código bem escrito, eficiente e compreensível.
- Clientes / Usuários: Tanto os comensais quanto os usuários finais são fundamentais, seja para desfrutar de uma refeição ou utilizar o software desenvolvido.

Para que essa engrenagem funcione, o time precisa trabalhar em conjunto e de forma harmônica. Cada indivíduo é responsável por uma parte da entrega do software. A seguir, abordei alguns papéis dentro da equipe para que você compreenda melhor suas funções. Também coletei depoimentos de profissionais da área que explicam de forma prática suas responsabilidades.

O papel do(a) **desenvolvedor(a)** é criar soluções de negócio com o uso da tecnologia, ou seja, desenvolver o software que resolverá determinado problema. O software é escrito ou desenvolvido por meio de linguagens de programação, que nada mais são do que uma coleção de programas e dados que instruem um computador como executar tarefas específicas.

"Atuar como um desenvolvedor é criar, projetar e manter processos reais através de soluções baseadas em sistemas de computador. Significa resolver problemas diversos através de soluções programáticas. Para resolver problemas complexos, um desenvolvedor utiliza uma ou mais linguagens de programação e

com elas compartilha um código-fonte com uma equipe onde testam e depuram seus códigos para que tudo funcione corretamente."

— Rodrigo Duclerc, Software Engineer na Nearsure

Quality Assurance (QA): seu trabalho é garantir qualidade da entrega do software. Após os desenvolvedores criarem determinadas funcionalidades ou correções do software, o QA é responsável por realizar as devidas validações e assegurar se o que será entregue atende aos requisitos dos usuários e cumpre suas funções.

"O papel do QA em um time de desenvolvimento de Software vai além de testar o software e encontrar defeitos. Seu trabalho está diretamente ligado a todas as etapas do processo de desenvolvimento, desde a concepção até a entrega do produto. O maior desafio dessa profissão é garantir que tanto o processo quanto o produto evoluam com qualidade, pois a cada versão entregue o software precisa continuar funcionando para não causar prejuízos para os usuários.

As principais habilidades desejáveis em um QA são: a facilidade de comunicação com todos os membros da equipe, a criatividade em descrever cenários de teste, a objetividade na descrição dos defeitos e a curiosidade ao buscar falhas no sistema."

— Anne Rocha, Senior Quality Assurance na Noesis

Muito se fala em experiência do usuário e o trabalho de **User Experience** (UX) vai além de fazer um design do produto. O foco está nos usuários finais que utilizarão o software, compreendendo suas necessidades e expectativas para criar produtos que proporcionem uma experiência eficiente, eficaz e satisfatória.

"Para atuar como UX, é necessário ter em mente que essa é uma área multidisciplinar, estratégica e com foco no ser humano. O grande desafio é entender a dor ou necessidade da pessoa usuária e propor a melhor solução possível com o tempo e recursos disponíveis, respeitando as regras do negócio. Para isso, é preciso

que o UX esteja alinhado às equipes de tecnologia e de negócios e coloque no centro das decisões os sentimentos e a experiência do consumidor final, sem deixar de lado a empatia pelo time de pessoas que está por trás do desenvolvimento."

— Dalila Azevedo, UX Designer na Globo

Esses são alguns dos papéis que contribuem para o desenvolvimento e a entrega do software. Claro, há outros envolvidos, como gerentes de projetos, (*DevOps*), analistas de negócio, entre outros, cada um com sua função essencial para o sucesso do produto final. Em resumo, o trabalho em equipe, a comunicação eficiente e o alinhamento das expectativas são fundamentais para que o software seja entregue com qualidade, atendendo às necessidades dos usuários e alcançando os objetivos do projeto.

Capítulo 2
Desenvolvedores não só escrevem códigos

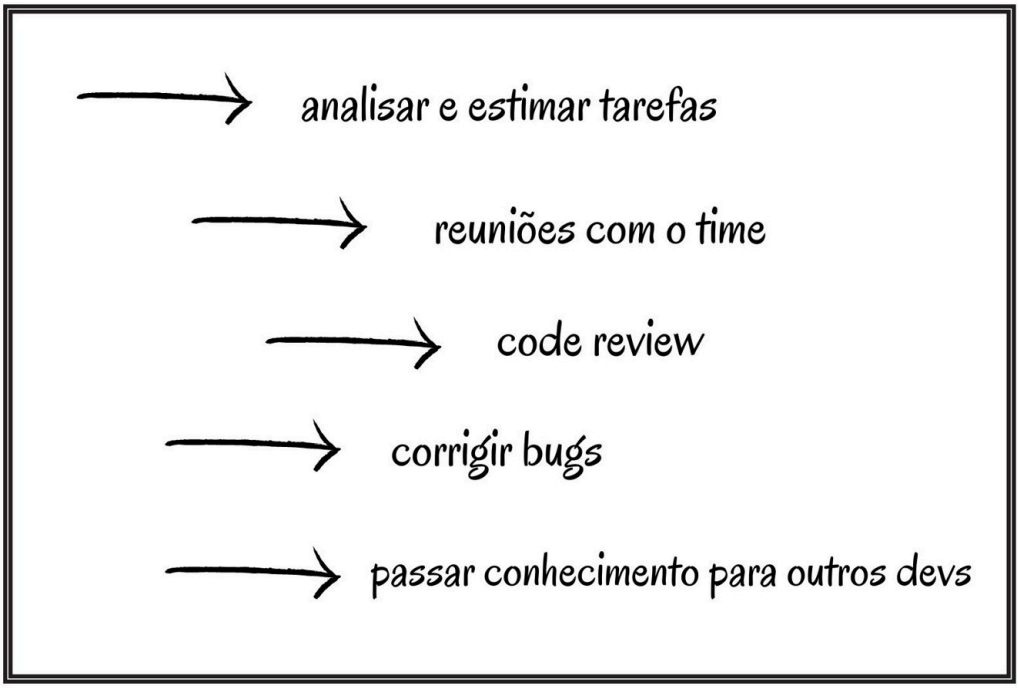

Figura 2.1: Tarefas do desenvolvedor.

Uma vez, fiz mentoria para uma moça que queria voltar para a área de TI após um intercâmbio que havia feito. Durante nossa conversa, ela me perguntou se programadores realmente codificavam o dia inteiro. Essa pergunta me fez refletir sobre a percepção comum da nossa profissão e o quão distante está da nossa realidade. Isso é ruim? Não, mas é curioso.

A verdade é que ser desenvolvedor(a) vai muito além de "simplesmente" escrever milhares de linhas de código. Envolve um conhecimento profundo sobre o negócio e participação ativa em todas as fases do ciclo do desenvolvimento do software, discutir soluções estratégicas para solucionar problemas, monitorar aplicações para evitar possíveis problemas, corrigir eventuais "bugs"

em ambiente de produção e participar de reuniões cerimoniais com o time de desenvolvimento.

Essa ideia de que "desenvolvedores não apenas codificam" não é algo que surgiu recentemente. O trabalho da pessoa desenvolvedora nunca foi só escrever código; é sobre entregar soluções para problemas de negócio utilizando a tecnologia como meio. Problemas no sistema sempre surgem; logo, sempre corrigimos bugs e falhas.

O que pude observar com a evolução da tecnologia e o surgimento de novos produtos e serviços é que os sistemas se tornaram cada vez mais complexos. A escala desses sistemas se tornou muito maior, assim como a quantidade de dados que geramos, a qual vem crescendo imensamente.

Com o aumento da complexidade dos sistemas, o papel dos desenvolvedores se expandiu significativamente. Já não são apenas codificadores isolados: tornaram-se solucionadores de problemas que entendem o sistema de forma profunda. Isso inclui não apenas o desenvolvimento, mas também a gestão do ciclo de vida completo do software — desde a concepção até a manutenção e o suporte pós-lançamento.

Essa expansão do escopo de trabalho dos desenvolvedores é o que levou ao surgimento do conceito de **Dev Full-Cycle** ou "desenvolvedor completo" — uma ideia que está se tornando cada vez mais relevante em empresas que lideram em inovação e eficiência operacional, como a Netflix. Greg Peters, CTO da Netflix, descreve essa abordagem em detalhes no Blog da Netflix. Ele destaca como essa integração de funções e responsabilidades é fundamental para o sucesso em um ambiente tecnológico que evolui rapidamente.

Em um artigo publicado no blog da Netflix *Full Cycle Developers at Netflix — Operate What You Build*, em 2018, Greg Peters explica como iniciou essa abordagem do conceito, após a Netflix ultrapassar 125 milhões de usuários e mais de 140 milhões de horas de

visualização por dia. Para atender a toda essa demanda, a equipe da Netflix tornou os desenvolvedores mais independentes dos times de operações, ou seja, eles podiam enviar seus códigos por conta própria para o ambiente de produção quando necessário, e também eram responsáveis por problemas de produção, monitoramento e observabilidade.

Essa mudança resultou em uma redução significativa da dependência dos desenvolvedores em relação ao time de operações. Traduzi um pequeno trecho do artigo que descreve uma prática comum em empresas que adotam uma cultura DevOps, em que se defende que os times de desenvolvimento de software sejam responsáveis não apenas pela construção das aplicações, mas também pela sua operação e manutenção após o lançamento.

"Ao combinar todas essas ideias, chegamos a um modelo em que uma equipe de desenvolvimento, equipada com incríveis ferramentas de produtividade para desenvolvedores, é responsável por todo o ciclo de vida do software: design, desenvolvimento, teste, implantação, operação e suporte.

Espera-se que os desenvolvedores de ciclo completo tenham conhecimento e sejam eficazes em todas as áreas do ciclo de vida do software. Para muitos desenvolvedores iniciantes na Netflix, isso significa avançar em áreas nas quais eles não se concentraram antes. Realizamos bootcamps de desenvolvimento e outras formas de treinamento contínuo para transmitir esse conhecimento e desenvolver essas habilidades. O conhecimento é necessário, mas não suficiente; ferramentas fáceis de usar para pipelines de implantação (por exemplo, Spinnaker) e monitoramento (por exemplo, Atlas) também são necessárias para uma propriedade eficaz do ciclo completo." (Netflix Tech Blog, 2018, traduzido de https://netflixtechblog.com/full-cycle-developers-at-netflix-a08c31f83249).

Como destacado, o papel dos desenvolvedores modernos é vasto e envolve uma variedade de responsabilidades que vão além de

escrever códigos. Desenvolvedores são solucionadores de problemas cujo trabalho impacta diretamente na operacionalidade e eficiência das aplicações que desenvolvem. Mas, mesmo preparados para lidar com todo o ciclo de vida do software, os desenvolvedores enfrentam desafios diários que interrompem seu fluxo de trabalho.

Se pararmos para analisar, percebemos que o tempo efetivo de codificação dos desenvolvedores não se estende por oito horas consecutivas devido a diversas interrupções e responsabilidades variadas. Além disso, como mencionado, existem outras tarefas que não envolvem somente codificação e, com isso, o tempo efetivo disponível para codificação é frequentemente menor do que o esperado.

A codificação, mesmo sendo uma parte crítica do software, raramente é uma atividade interruptiva. Desenvolvedores enfrentam múltiplas interrupções ao longo do dia, que podem ocorrer de várias formas. Isso inclui problemas emergenciais que necessitam de uma atenção imediata, reuniões não agendadas ou auxílio a colegas com tarefas — seja oferecendo suporte ou retirando um bloqueio.

Essas interrupções são comuns e, muitas vezes, necessárias. Elas exigem uma transição mental, uma troca de contexto que pode impactar na produtividade. Cada vez que um desenvolvedor para e retorna à sua tarefa original, leva um tempo até retomar o raciocínio daquele contexto que havia sido interrompido.

Um artigo publicado no site da Software destaca uma estatística interessante. A pesquisa foi realizada com base em 250 mil desenvolvedores da comunidade global e foi identificado que os desenvolvedores codificam 52 minutos por dia, cerca de 4 horas e 21 minutos durante uma semana normal de trabalho.

A avaliação chegou à seguinte conclusão:

"Nossa análise revelou que cerca de 45% de toda a codificação do dia de trabalho ocorre entre as 14h e as 17h, indicando que os desenvolvedores podem estar mais ocupados com reuniões e outros

trabalhos durante a manhã. A codificação tende a começar no final do dia e continuar após o horário tradicional de trabalho. Apenas 10% da codificação ocorre entre 9h e 11h, enquanto 12% da codificação ocorre após o horário de trabalho, entre 17h e 19h." Mason McLead, Chief Technology Officer da Software, outubro de 2021. (Software Reports Code Time, 2021, traduzido de https://www.Software.com/reports/code-time-report).

Figura 2.2: Média de tempo que o desenvolvedor codifica.

Embora a pesquisa seja de 2021, acredito que pouca coisa tenha mudado desde então. Esta realidade continua sendo a de muitos desenvolvedores, que dividem seu tempo em múltiplas tarefas que vão muito além da codificação pura. Mas uma coisa é clara: é muito incomum ver desenvolvedores codificando durante toda uma jornada de oito horas sem qualquer interrupção.

Obtive depoimentos de profissionais da área que relataram seus pontos de vista sobre o papel do desenvolvedor atual, as

interrupções e outras tarefas além da codificação.

"Atualmente os desenvolvedores de software precisam de mais do que habilidades técnicas (hard skills); precisam ir além de 'apenas' escrever códigos. O desenvolvedor deve ter conhecimento mais aprofundado do negócio/produto e mais contato com o cliente, visto que essa proximidade vai trazer um maior entendimento da demanda, e, consequentemente, em muitos casos, trará uma visão diferente daquela do Product Owner, fazendo com que a discussão seja mais rica e proveitosa."

— Denis Souza, Tech Lead na GFT Technologies

"Ser desenvolvedor vai além de escrever código. É entender problemas, se comunicar bem, encontrar soluções de problemas e colaborar com a equipe. Não somos só programadores, somos solucionadores. Nossas habilidades de transformar complexidade em simplicidade impulsionam a inovação e a criação de valor para usuários finais."

— Khaled Al Jadaan, Senior Software Engineer no 4SolutionGroup

"Essa descoberta parece capturar um aspecto humano universal. Quem de nós não se viu, pela manhã, lutando contra o sono e o desejo de um café forte? Essa pesquisa sugere que nós, desenvolvedores, muitas vezes, usamos as manhãs para ponderar ideias com colegas, discutir estratégias e traçar o rumo do dia. É como se estivéssemos ajustando nosso ritmo para o dia que se inicia.

E então, algo mágico acontece no coração da tarde. O ambiente ao nosso redor parece relaxar, as interrupções diminuem e encontramos nosso fluxo. É quando as linhas de código começam a se encaixar como peças de um quebra-cabeça, e o mundo digital toma forma sob nossos dedos. É quase como se as criações digitais ganhassem vida durante essas horas.

E o que dizer das tardes estendidas, quando muitos de nós optam por continuar codificando além do horário "convencional" de trabalho? Isso ressalta nosso desejo inato de explorar, criar e mergulhar profundamente em nossos projetos. Muitas vezes, é nesses momentos que as ideias mais brilhantes surgem, quando não há pressa ou restrições temporais.

No entanto, é interessante observar como essa dinâmica se manifesta na Europa. Com uma apreciação profundamente enraizada no equilíbrio entre vida profissional e pessoal, muitos países europeus abraçam horários de trabalho flexíveis. Isso pode influenciar a distribuição da codificação ao longo do dia, priorizando um equilíbrio saudável.

Essa pesquisa nos lembra que somos mais do que meros digitadores de código. Somos seres humanos complexos, com ritmos naturais e fluxos criativos. Ao abraçar e entender esses ritmos, não apenas nos tornamos desenvolvedores mais realizados, mas também seres humanos mais bem sintonizados com nosso próprio potencial e com as nuances das culturas ao nosso redor."

— Daniel Jesus, Engineer Manager na XP

"Inicialmente, os desenvolvedores dedicam mais tempo a resolver problemas técnicos do que à codificação em si, o que pode parecer que estão sempre codificando. Com a experiência, a ênfase muda para entender como aplicar a tecnologia a desafios complexos. Isso envolve análises, tomadas de decisão, reuniões e colaboração em várias etapas. Desenvolvedores seniores traduzem as necessidades dos clientes em soluções tecnológicas, considerando usabilidade, segurança e escalabilidade.

Assim, o tempo de codificação direta pode diminuir, não por acaso, mas porque estão envolvidos em todas as fases do desenvolvimento. O papel evolui para algo estratégico e abrangente, não apenas codificação."

— Matheus Catossi, CTO na System Frame

"Os desenvolvedores não passam a maior parte do seu tempo codificando. Existem outras atividades que vêm antes e fazem toda a diferença na hora de botar 'a mão na massa'. Já gastei horas para descobrir que um ponto e vírgula resolveria um problema, ou até mesmo um mês para entender como alterar um relatório.

Precisei dedicar horas em análise e reuniões para ter sucesso nestes desafios. Engana-se quem acha que programadores colocam seus fones de ouvido e se isolam do mundo o dia inteiro."

— Tiago Aguiar, Senior Software Engineer na InfoJobs

"Eu acredito que essa é uma verdade. Pessoalmente falando, vejo que a maioria dos momentos que eu estou codando tendem a acontecer depois das 15h. Porém, eu evito ao máximo continuar após meu horário definido de trabalho, e isso deveria ser evitado pela maioria das pessoas também.

Atualmente, com o avanço do trabalho remoto, estamos cada vez mais sobrecarregados com reuniões. Quanto mais sênior você for, mais reuniões participará. A realidade é que não existe um caminho em desenvolvimento que lhe permita apenas programar; uma hora ou outra você vai acabar tendo que gerenciar times e participar de reuniões. A questão é saber quando uma reunião é produtiva ou não, caso contrário, você acabará desperdiçando tempo em vez de fazer algo produtivo."

— Lucas Santos, Specialist Engineer na Hemnet e Educador Profissional.

"É intrigante considerar que os programadores, em média, dedicam somente 52 minutos por dia à programação. A minha própria experiência, embora tenha abrangido várias empresas e dinâmicas distintas, contradiz essa realidade. Em um ambiente de trabalho com um time composto por pelo menos três indivíduos (afinal, dois não formam um time, apenas uma dupla, não é mesmo?), é plausível imaginar uma distribuição mais equitativa das responsabilidades. Por exemplo, um profissional júnior investe a maior parte do seu tempo

– cerca de 5 horas diárias – em programação direta. Simultaneamente, um programador pleno, contribuindo com aproximadamente 3 horas de programação diária, oferece suporte ao júnior e enriquece o desenvolvimento do projeto.

Em relação ao programador sênior, a situação pode variar. Dada a quantidade de tarefas de liderança, a codificação direta pode não ser sua prioridade. No entanto, sua função de orientar os outros membros da equipe e compartilhar conhecimentos é inestimável. Mesmo considerando esse cenário, ainda assim atingimos uma média de 6 horas diárias de codificação, o que contrasta fortemente com os 52 minutos propostos.

Para além da experiência pessoal, o mercado de trabalho também lança dúvidas sobre essa alegação. Ao traçar um panorama mais amplo, é notório que muitos projetos demandam um esforço maior do que o sugerido. A complexidade das tarefas, os prazos apertados e a necessidade de colaboração eficaz entre os membros da equipe impulsionam a dedicação substancial à programação.

Assim, ao somar a evidência da minha trajetória profissional com a compreensão da dinâmica do mercado, é difícil concordar com a ideia de que a média de 52 minutos de programação por dia seja condizente com a realidade. A programação é o cerne da atividade dos programadores, e a sua magnitude é condizente com a natureza desafiadora e envolvente desse campo."

— Italo Oliveira, Senior Software Engineer na Meteor.Js

"É de suma importância que o time de engenharia esteja muito bem alinhado com o time de produto e/ou de negócio. Um planejamento bem estruturado é meio caminho andado para o sucesso da entrega, e é por isso que hoje, mais do que nunca, a pessoa desenvolvedora deve estar muito ciente do valor que será entregue ao cliente final. Para isso, é importante que haja todo um processo que incentive práticas de Discovery, criação de POCs, testes de usabilidade etc.

Sendo assim, nota-se que escrever código demanda apenas uma pequena parcela do tempo.

O restante — que é o mais crucial — é destinado ao entendimento sobre qual feature vai impactar mais nas necessidades do cliente. Empatia, resiliência e criatividade são fatores decisivos para alcançar esse objetivo. Mas já adianto: não é algo fácil de conseguir."

— Angelo Belchior, Staff Engineer na PicPay

Durante a conversa que tive com esses profissionais em relação a este artigo, a maioria possui experiências semelhantes e sabe que há essas interrupções. Portanto, concorda com a escrita do texto.

No próximo capítulo, apresentarei um pouco sobre as metodologias ágeis. Elas proporcionam uma estrutura que pode ajudar as equipes a lidar com a incerteza e a complexidade típicas dos projetos modernos de desenvolvimento de software. É importante que desenvolvedores conheçam esses modelos, pois precisamos estar adaptados às mudanças e entregar valor de forma consistente para o negócio.

Capítulo 3
Metodologias Ágeis

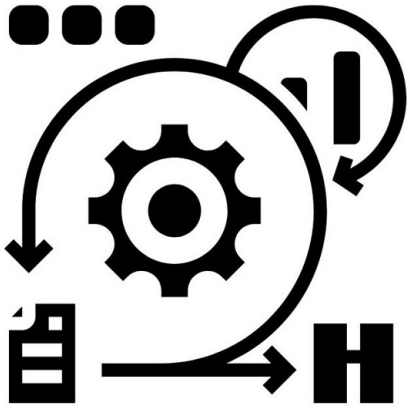

Figura 3.1: Metodologia Ágil.

O mundo do desenvolvimento de software é muito dinâmico, e à medida que os projetos se tornaram mais complexos e as demandas do mercado mais rápidas, as metodologias antigas já não supriam as necessidades, eram rígidas e engessadas. Foi através desse contexto que as metodologias ágeis como Extreme Programming (XP) e o Scrum ganharam destaque.

Metodologia ágil não significa apenas fazer mais e mais rápido. Trata-se de como as equipes podem trabalhar de maneira mais eficiente, adaptável e alinhada aos objetivos do negócio. A agilidade está voltada ao processo de adaptação, permitindo que você responda rapidamente às mudanças e satisfaça o cliente com entregas contínuas.

Em times ágeis, espera-se que a equipe, como um todo, questione, participe ativamente das cerimônias e dos alinhamentos e aponte os aspectos negativos e positivos da Sprint, contribuindo para possíveis melhorias futuras.

Extreme Programming (XP)

Extreme Programming (XP) é uma metodologia ágil de desenvolvimento de software criada na década de 1990 por Kent Beck. É uma metodologia bem-sucedida porque enfatiza a satisfação do cliente. Em vez de entregar tudo o que você poderia desejar em alguma data no futuro, esse processo entrega o software que você precisa, conforme necessário. Extreme Programming capacita seus desenvolvedores a responder com confiança às mudanças nos requisitos dos clientes, mesmo no final do ciclo de vida.

Segundo o site *extremeprogramming.org*, essa metodologia melhora um projeto de software de quatro maneiras essenciais: comunicação contínua, simplicidade, feedback constante e coragem.

- Comunicação contínua: facilita o compartilhamento do conhecimento entre membros do time e mantém uma comunicação constante com o cliente.
- Simplicidade: Foca no que é apenas necessário, evita complexidades desnecessárias ao se deparar com novos problemas.
- Feedback constante: incentiva pequenas entregas frequentes para que o retorno do cliente seja rápido e incorporado ao processo.
- Coragem: encoraja a equipe a aceitar mudanças de requisitos mesmo durante o estágio de desenvolvimento, para que o time esteja disposto a fazer refatorações assim que necessário.

Os Extreme Programmers mantêm a comunicação constante com seus clientes e colegas programadores. Na prática, o software é desenvolvido modularmente.

O processo funciona da seguinte forma: o módulo é desenvolvido pelo time de desenvolvimento, depois é testado e, em seguida, validado pelo cliente que fornece um rápido feedback. Caso sejam necessárias implementações ou modificações, as alterações são enviadas de volta ao time de desenvolvimento para ajustes.

Scrum

Scrum é uma metodologia ágil para gestão e planejamento de projetos de software. De acordo com o *Guia do Scrum*, escrito por Ken Schwaber e Jeff Sutherland, o Scrum é definido como um framework para desenvolver e manter produtos complexos. Esse framework estrutural serve como um conjunto de boas práticas que auxiliam no gerenciamento eficaz de projetos, promovendo uma abordagem iterativa e incremental. Isso facilita a entrega de produtos de alta qualidade de maneira consistente.

No Scrum, os projetos são divididos em ciclos chamados de Sprints, que geralmente duram de duas a quatro semanas. Durante cada ciclo, uma nova funcionalidade é desenvolvida e entregue em partes, seguindo um processo de planejamento, execução, revisão e adaptação. O progresso do projeto é revisado e monitorado em reuniões diárias, conhecidas como *daily scrum*. Essas reuniões são conduzidas pelo Scrum Master, juntamente da equipe, que é autogerenciável e auto-organizável.

A entrega do produto é de forma incremental, o que pode maximizar oportunidades de realimentação e sempre priorizar e mitigar riscos. A seguir estão os três principais papéis do Scrum e sua importância:

- **Product Owner:** É a pessoa responsável pelo negócio e representa os interesses dos stakeholders. O Product Owner define as funcionalidades do produto e prioriza os itens do backlog. Um de seus principais papéis é garantir que o backlog do produto seja visível e transparente para todos, para que o time de desenvolvimento entenda os itens e suas prioridades.
- **Scrum Master:** É a pessoa responsável por garantir que o Scrum seja aplicado da maneira correta. Normalmente, é a pessoa que fica responsável pelas cerimônias, ajudando o time a lidar com impedimentos externos ao Time Scrum.

- **Equipe de Desenvolvimento:** É composta por todos os tipos de pessoas, incluindo desenvolvedores, engenheiros de qualidade, gerentes etc.

Os eventos ou cerimônias do Scrum visam criar rotinas e minimizar as necessidades de reuniões não definidas no Scrum. Esses eventos têm uma duração de tempo estipulada para que o time possa focar nos objetivos de cada reunião, sendo diretos e não perdendo o foco para atingir determinado propósito.

- **Retrospectiva:** São reuniões essenciais para melhoria contínua. Permite que a equipe reflita sobre o que funcionou bem e o que poderia ser melhorado, além de considerar lições aprendidas e casos que surgirem.
- **Sprint Review:** É a oportunidade que o time tem para apresentar aos stakeholders o trabalho realizado durante a Sprint, possibilitando feedbacks e sugestões conforme necessário.
- **Planejamento de Sprint:** Momento em que a equipe define o escopo do trabalho para a próxima Sprint, baseando-se nas prioridades do negócio e na capacidade do time.
- **Daily Scrum:** Encontro diário em que cada membro do time informa em qual tarefa está atuando, o progresso, se há algum impedimento e o que foi feito no dia anterior.
- **Refinamento técnico:** Permite que o time analise e refine as "histórias da Sprint" de maneira mais detalhada. As histórias da Sprint são conjuntos de tarefas ou requisitos que o time se compromete a completar durante uma Sprint. Neste processo, a equipe pode subdividir essas histórias em tarefas menores conhecidas como *tasks*. O refinamento envolve também a estimativa de tempo necessária para conclusão das atividades,

certificando que tudo seja entregue dentro do prazo estabelecido para Sprint.

As principais diferenças entre XP e Scrum

As duas metodologias, Scrum e XP (Extreme Programming), são as mais populares no desenvolvimento ágil de software e compartilham o objetivo de entregar software de qualidade e de forma eficiente, valorizando as interações e os feedbacks. Entretanto, existem algumas diferenças importantes entre as duas, que são:

- Foco: Scrum se concentra na gestão do projeto e na entrega iterativa de produtos. Já Extreme Programming (XP), por sua vez, enfatiza a qualidade de código através de práticas de engenharia, como a programação em pares e o desenvolvimento orientado a testes, o famoso TDD (*Test Driven Development*).
- Entregas: Tanto o Scrum quanto o XP promovem entregas iterativas. No Scrum, essas entregas são feitas ao final de cada Sprint (que pode durar de duas a quatro semanas).
- Papéis: O Scrum tem seus papéis claramente definidos, como Scrum Master, Product Owner e a Equipe de Desenvolvimento. Por outro lado, no Extreme Programming, embora os papéis não sejam rigidamente definidos, as responsabilidades são compartilhadas, permitindo maior flexibilidade em relação a quem executa cada tarefa dentro da equipe.

Durante meu tempo de trabalho na Globo, tive a oportunidade de trabalhar com a Luanna Peixoto, que atuava como PO (*Product Owner*) no time de Produção de Conteúdo. Pedi a ela que compartilhasse um pouco sobre seu trabalho diário. Achei valioso incluir a visão de uma profissional sobre as responsabilidades e desafios enfrentados por uma PO no dia a dia. Considero a Luanna

Peixoto uma excelente PO. Além de ter forte domínio sobre o produto, ela possui conhecimentos técnicos, sabe equilibrar e negociar as entregas.

"O principal pilar do PO é a comunicação. Por meio dela, estabelece-se o elo entre as pessoas e suas necessidades. Com uma visão de futuro — orientada a objetivos corporativos e métricas — e foco no que gera mais impacto positivo no menor tempo, a pessoa PO torna-se responsável por orquestrar o consenso, validar hipóteses e facilitar a priorização junto aos stakeholders, mantendo todos os envolvidos alinhados, gerindo expectativas e reduzindo riscos. Dessa forma, sabe adaptar e incrementar possíveis necessidades.

Outro grande pilar é a empatia, aliada à escuta ativa e à disponibilidade. Essas são ferramentas importantes para o engajamento do time multidisciplinar, área de negócio e usuários finais, permitindo o entendimento do propósito e, consequentemente, a obtenção do melhor resultado final."

— Luanna Peixoto, coordenadora de Soluções e Tecnologia na Globo

A escolha de qual metodologia ágil adotar dependerá das necessidades específicas do projeto e de como é feita a dinâmica do time.

CAPÍTULO 4
De Junior a Sênior: erros, acertos e a paciência para escalada do sucesso

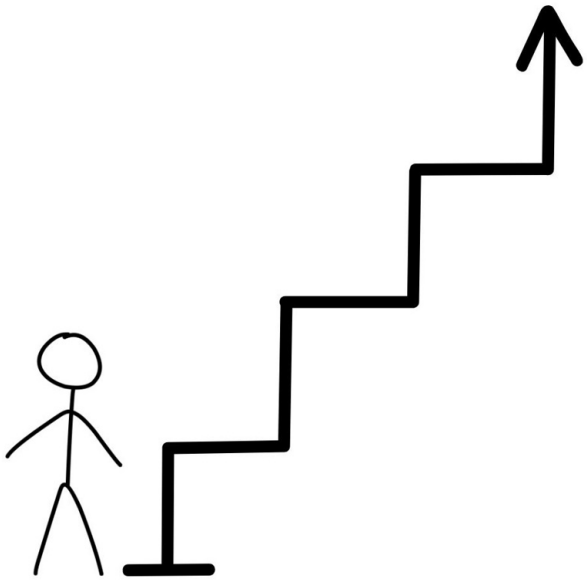

Figura 4.1: Degraus de carreira.

Recordo-me de meu primeiro estágio em desenvolvimento de software. Naquela época, as coisas eram mais lentas se comparadas aos dias de hoje. Durante minha formação acadêmica, utilizávamos a linguagem Java e, para praticar a lógica de programação, recorríamos ao Portugol, escrevendo no caderno o fluxo dos dados e as estruturas de controle do programa.

Os recursos eram limitados; palestras em minha instituição eram raridades. Meus principais meios de aprendizado incluíam livros e alguns blogs com grande relevância na época, todos descobertos através do motor de buscas do Google.

A web tornou-se mais colaborativa com o passar do tempo. O avanço tecnológico e o surgimento de novas ferramentas possibilitaram que mais pessoas contribuíssem com conteúdos. O

acesso ao conhecimento ampliou-se drasticamente com a quantidade de cursos e conteúdos disponíveis, tornando muito mais fácil obter informações e soluções rapidamente.

Dispor de tantos materiais é uma bênção, não é? Mas de que adianta termos tantos materiais disponíveis para aprender se não temos foco naquilo a que nos dedicar? E, com as mudanças desenfreadas da tecnologia, somos meio que obrigados a nos atualizar, a estudar constantemente. O desejo ou a necessidade de aprender algo novo, especialmente para conquistar uma vaga específica ou para subir de nível na carreira, pode gerar uma ansiedade significativa.

Dentro da carreira de desenvolvimento de software, existem diferentes categorias que definem o nível de habilidade e responsabilidade de um profissional — como Júnior, Pleno e Sênior. Essa definição refere-se ao grau de maturidade, experiência e responsabilidade que uma pessoa desenvolvedora possui ou deve alcançar ao longo da carreira.

Na sequência, vamos detalhar mais esses níveis.

Desenvolvedor(a) Júnior

Geralmente, esse título é atribuído à pessoa que acabou de sair da faculdade e está dando os primeiros passos na carreira de desenvolvimento, já tendo concluído a etapa de estágio — ou não. O(a) desenvolvedor(a) iniciante precisa ser mentorado(a) por alguém mais experiente da área, pois ainda está em processo de aprendizado constante e, por isso, precisa do apoio de pessoas mais experientes.

É natural que seu código precise ser revisado, que suas tarefas precisem ser segmentadas e avaliadas por outros profissionais para facilitar sua execução.

Não se pode cobrar de um desenvolvedor júnior conhecimentos arquiteturais nem a mesma produtividade de uma pessoa desenvolvedora mais experiente no mercado.

Desenvolvedor(a) Pleno

Este é o nível intermediário da carreira, quando o(a) desenvolvedor(a) já superou a fase inicial de aprendizado e possui um conhecimento técnico mais consolidado.

O(a) desenvolvedor(a) pleno(a) ainda está em busca de aprimoramento técnico para evoluir em direção à senioridade, mas é capaz de trabalhar com mais autonomia, lidar com tarefas mais complexas e contribuir frequentemente para a tomada de decisões técnicas.

Além disso, espera-se que o desenvolvedor pleno atue como mentor de desenvolvedores juniores, auxiliando no crescimento da equipe.

Desenvolvedor(a) Sênior

Este é um dos níveis mais avançados da carreira de desenvolvimento de software.

O(a) desenvolvedor(a) sênior não é reconhecido apenas por sua longa experiência com tecnologias, práticas e mercado, mas também por demonstrar um profundo entendimento dos processos de negócio, das estratégias e dos objetivos organizacionais.

Sua experiência se traduz em habilidades altamente eficazes de resolução de problemas, capacidade de projetar sistemas robustos e escaláveis, além de aptidão para mentorar membros menos experientes da equipe.

Tech Lead

O papel do Tech Lead, como o próprio nome sugere, é ser líder técnico do time. Essa pessoa é responsável por elaborar a

arquitetura do projeto, sempre com foco nos objetivos do negócio, e por oferecer suporte aos desenvolvedores de todos os níveis, desde iniciantes até sêniores.

O líder técnico geralmente interage com o time de arquitetura, comunicando mudanças arquiteturais ao restante da equipe, resolvendo impedimentos que possam surgir e, muitas vezes, é a ele que os sêniores recorrem quando há alguma dúvida ou bloqueio relacionado ao negócio.

Arquiteto(a) de Software

O Arquiteto de Software é responsável pelo desenho de toda a arquitetura do projeto. Suas principais atividades incluem criar estratégias para o desenvolvimento de aplicações escaláveis e tolerantes a falhas, que atendam às demandas do negócio.

Além disso, o arquiteto é responsável pela escolha das ferramentas que serão usadas no projeto, atentando-se sempre aos custos e ao impacto que terão no resultado final.

Crescimento e maturidade na carreira de desenvolvimento

E não para por aí: há outros níveis hierárquicos acima do Arquiteto de Software que não serão detalhados aqui. A jornada da escada do crescimento profissional vai além da aquisição de títulos ou posições específicas e reflete a importância de uma evolução constante no campo de desenvolvimento de software. Por isso, é preciso ter paciência e aproveitar cada degrau em que está subindo.

Naturalmente, ao considerar cada nível da carreira de desenvolvimento, espera-se um certo período de experiência e maturidade associados. É importante que a pessoa desenvolvedora tenha essa experiência para transitar por cada etapa, para absorver não apenas o tempo necessário e o conhecimento técnico, mas

também a maturidade profissional. Portanto, é preciso ter paciência para adquirir essas habilidades e maturidade que só vêm com a prática e o decorrer do tempo. E, claro, também não é preciso pular etapas.

Os erros e "feridas" que adquirimos durante nossa jornada são aprendizados que jamais esqueceremos. Costumo dizer que é preciso "apanhar" um pouco para aprender — aprender, primeiro, pelo caminho mais difícil. Enfrentar situações desafiadoras ou até mesmo dolorosas pode proporcionar lições valiosas que contribuem para o crescimento e a resiliência — algo dificilmente replicável apenas por meio de livros ou cursos.

Além disso, é comum encontrar desenvolvedores sêniores que permanecem por muitos anos na mesma empresa, o que pode refletir a maturidade e a estabilidade alcançadas em relação ao objetivo de continuar crescendo dentro da organização.

Permanecer durante alguns anos na mesma empresa permite que a pessoa desenvolvedora aprofunde seus conhecimentos sobre o negócio e a estrutura organizacional como um todo, sendo capaz de mentorar novos desenvolvedores iniciantes e de propor soluções que impactam positivamente a corporação.

Recentemente, observei uma certa ansiedade entre os desenvolvedores iniciantes que estão começando na área. Percebi um impulso em aprender tudo rapidamente para subir o nível de carreira. Só que isso, muitas vezes, leva ao esquecimento da importância de vivenciar cada momento da trajetória profissional. O entusiasmo e o desejo de crescer podem levar à busca precipitada por promoções ou por mudanças frequentes de emprego.

É importante equilibrar essa ambição com a necessidade de adquirir experiência prática e de realmente desenvolver habilidades essenciais. Hoje em dia, tornou-se uma tendência desvalorizar o tempo de formação sólida — como uma graduação de quatro anos,

por exemplo — em favor de cursos rápidos, que prometem uma entrada "fácil" no mercado de trabalho com salários promissores.

Você pode optar por fazer esses cursos, mas precisará estar ciente de que será necessário um esforço adicional para buscar conhecimentos não abordados durante o curso, bem como buscar a experiência prática necessária para, de fato, dominá-los.

Entrevistei alguns desenvolvedores e líderes técnicos para entender suas opiniões sobre a jornada do desenvolvedor e a importância de não pular etapas na carreira.

"Para aqueles que entendem que a paciência é o alicerce sólido, tanto na construção de linhas de código bem estruturadas quanto no trato sensível com as pessoas, a jornada no mundo do desenvolvimento de soluções será um tanto quanto tranquila, podendo ser até rápida, e cada degrau rumo ao sucesso terá seu significado."

— Henrique Eduardo Souza, Architect Manager na Nio

"Ninguém é igual a ninguém. Você talvez não seja tão bom quanto acredita, mas também não é tão ruim quanto pensa. Evite comparar-se constantemente com os outros. Você não sabe quais desafios o 'vizinho' enfrentou para chegar aonde está. A sorte desempenha um papel, e alguns podem ter mais do que outros. Concentre-se em seus próprios esforços, não seja covarde e desonesto e permita-se arriscar. Não confunda pressa com rapidez. Seja feliz e evite perder tempo gastando energia com coisas desnecessárias."

— Daniel Coelho, Tech Lead na Transfero Group

"Em um mundo cada vez mais acelerado e cheio de pessoas prometendo transformar você em um sênior em 6 meses com altos salários, é importante saber que a realidade não é bem assim, e cada um tem a sua própria velocidade. Não se compare a ninguém a

não ser com você mesmo. Ter paciência com persistência é fundamental em nossa área.

Como a área de TI é muito ampla, acredito que os novos desenvolvedores ficam muito perdidos ainda em saber qual direção seguir — tanto sobre o que estudar como para chegar a um nível maior na carreira. Somos diariamente bombardeados de informações, e, se isso for usado de forma indevida, pode atrapalhar e gerar mais indecisão. Comece com um passo de cada vez, entenda os fundamentos primeiro e depois siga com as tecnologias. Lembre-se de que tudo são só ferramentas. O mais importante é ter uma boa base de conhecimento e ter objetivos claros de curto e médio prazo."

— Flavio Francisco, Senior Software Developer na NIOZ Royal Netherlands Institute for Sea Research

"O mercado passou por inúmeras mudanças desde que entrei na área em 2009. No começo, não existia o cargo "fullstack" porque era esperado que você, desde iniciante, soubesse como funciona toda a esteira de desenvolvimento e entrega. As necessidades do mercado abriram novas posições e cargos para suprir as demandas em aberto.

Só é possível acelerar a carreira se você se propõe a enfrentar o máximo de desafios possíveis em menos tempo. Não há evolução sem risco. É preciso testar novas tecnologias e novas abordagens.

A coisa mais importante que pessoas desenvolvedoras precisam absorver é que entender do negócio é crucial. O meio de resolução do problema importa muito, mas só é possível resolver problemas que entendemos, pois, nesses casos, tentativa e erro podem custar caro para as empresas. Entregar código fácil de manter e de monitorar é a chave do sucesso para a carreira"

— Maurício Antunes, Senior Software Engineer na Clerk.com

"No papel de liderança técnica na Platform Builders, desempenho diversas atribuições, incluindo orientar e direcionar o time de desenvolvimento durante a execução dos projetos. Uma das minhas principais responsabilidades é apresentar ao time de desenvolvimento as demandas solicitadas pela equipe de negócios. Esse processo envolve discutir, em conjunto, a solução que será criada para atender às necessidades identificadas."

Nessa etapa, é essencial que o líder técnico esteja munido de todas as informações relevantes e seja capaz de transmitir essas informações de forma clara e objetiva. Isso garante que todos os membros da equipe compreendam plenamente o que está sendo solicitado pela área de negócios, facilitando a colaboração e a criação de soluções eficazes.

Outra atribuição importante no meu papel de liderança técnica é garantir que todos os membros da equipe estejam alinhados na mesma direção. Para isso, é fundamental um bom gerenciamento do processo de desenvolvimento, sustentado por boas práticas, como:

- *Escrita de testes unitários;*
- *Gestão de branches utilizando estratégias como o Gitflow;*
- *Uso de ferramentas de análise de qualidade de código;*
- *Promoção da comunicação e colaboração entre os membros da equipe por meio de metodologias ágeis.*

Cabe ao líder técnico promover e assegurar a aplicação consistente dessas práticas, pois elas são essenciais para alcançar entregas de alta qualidade. Além disso, elas ajudam a manter o alinhamento do time e a criar um ambiente de trabalho eficiente e colaborativo.

Por fim, uma atribuição essencial no meu dia a dia é a gestão de pessoas. Esse aspecto envolve diversas responsabilidades, como fornecer feedbacks construtivos, analisar o desempenho de cada membro da equipe, compartilhar conhecimentos e contribuir para o aprendizado e desenvolvimento individual. Além disso, é meu papel criar e manter um ambiente de trabalho saudável, onde todos se

sintam valorizados e motivados a crescer tanto profissional quanto pessoalmente.”

— Reginaldo Barros, Tech Lead na Platform Builders

Refletindo sobre as opiniões dos profissionais entrevistados, percebo como o cenário de desenvolvimento de software evoluiu e continua a mudar desde o início da minha carreira. Como o Flávio mencionou, o acúmulo de informações acaba atrapalhando os novos iniciantes, que muitas vezes se sentem perdidos com tantas opções de estudo, sem um direcionamento claro sobre por onde começar ou o que priorizar.

Portanto, seja pela urgência dos novos desenvolvedores em avançar rapidamente ou pela sabedoria dos mais experientes em valorizar cada etapa do aprendizado, a jornada no desenvolvimento de software é um mundo rico de experiências e oportunidades de aprendizado.

Cada um escolhe seu próprio caminho, mas sempre fundamentado na constante evolução e adaptação. É aconselhável manter a calma e a paciência para não pular degraus na escada do sucesso profissional.

Parte II: 10 hábitos que te tornarão um(a) desenvolvedor(a) eficiente

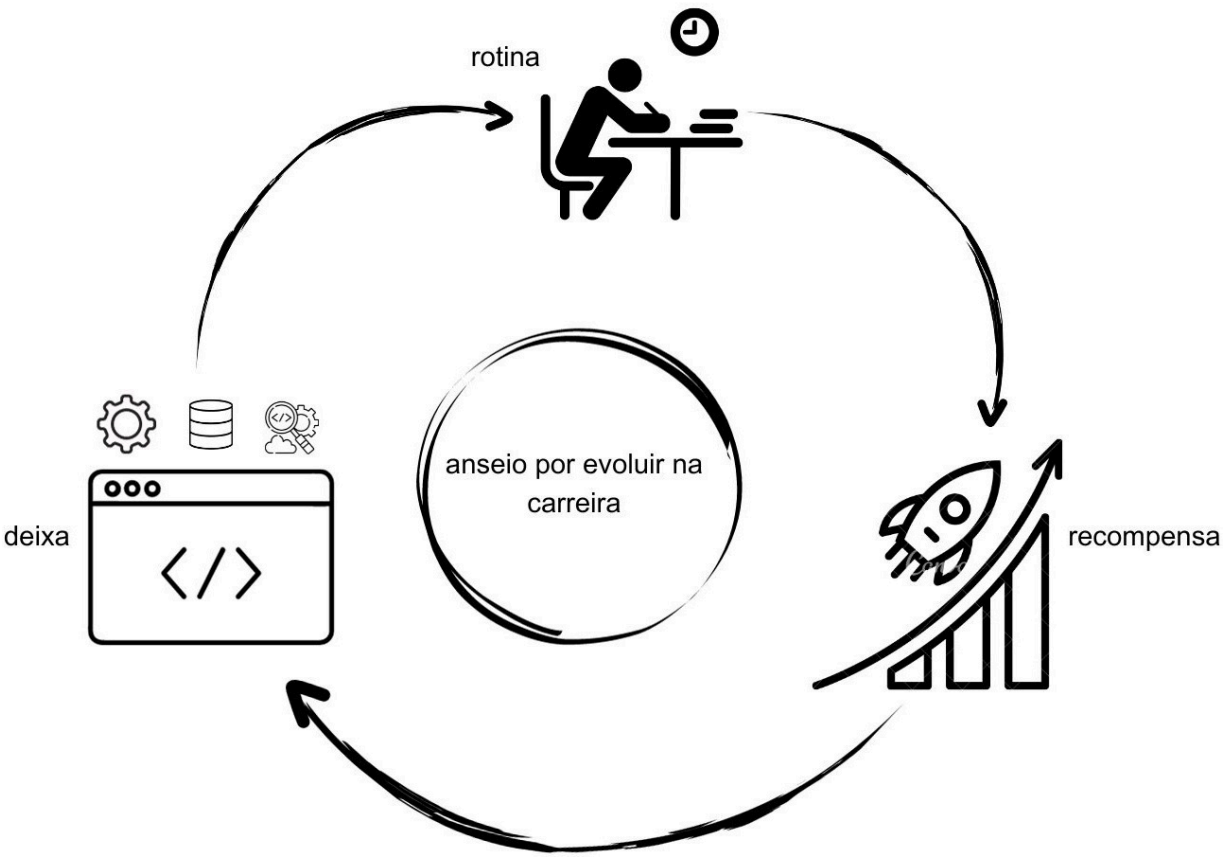

Figura 1: Hábitos de estudo.

Capítulo 5
O hábito

Antes de compartilhar algumas dicas de hábitos que o ajudarão a tornar-se um(a) desenvolvedor(a) de software eficiente, é importante compreender o que são hábitos e como eles funcionam.

Hábitos são comportamentos, pensamentos ou emoções que adquirimos e que se repetem quase automaticamente. Eles podem ser positivos, como praticar alguma atividade física, reduzir o consumo de alimentos que não fazem bem para você ou escovar os dentes. Também há hábitos negativos, como consumir frequentemente alimentos nocivos ao corpo ou perder noites de sono passando muito tempo no *smartphone*, navegando em redes sociais.

Charles Duhigg, em seu livro *"O Poder do Hábito"* (2012), explora como os hábitos influenciam nossas ações e determinam nosso estilo de vida. Além disso, a obra oferece uma compreensão valiosa sobre como podemos mudá-los. O autor descreve o processo de formação de um novo hábito, destacando que a maioria de nossas ações ocorre no "piloto automático". Para mudar um hábito, é preciso compreender o estímulo que desencadeia a ação, o comportamento (rotina) e, por fim, a recompensa satisfatória que o cérebro entende como dever cumprido.

Assim como os hábitos transformam nosso estilo de vida, no ambiente profissional eles também desempenham um papel importante: tornam-se essenciais para alcançar objetivos e obter melhores resultados, seja ao aprender algo novo ou aprimorar alguma habilidade existente. Na área tecnológica, onde o aprendizado contínuo é fundamental, cultivar boas práticas e manter a constância pode transformar você em um(a) desenvolvedor(a) mais eficiente.

Ao longo desta jornada, compartilharei hábitos e dicas que contribuirão para seu desenvolvimento profissional, com base na minha experiência e na de outros excelentes profissionais. Espero que, ao adotá-los, você alcance uma evolução significativa em suas habilidades técnicas e não técnicas, refletindo positivamente em seu ambiente de trabalho.

Após me interessar pelo assunto de hábitos, comecei a buscar conteúdos e livros do mesmo tema para ler. Recordo-me de que

recomendaram a biografia de Kobe Bryant, cuja estratégia para alcançar resultados achei genial.

Kobe foi um jogador profissional de basquetebol estadunidense e jogou, durante toda sua carreira, como ala-armador no Los Angeles Lakers, da National Basketball Association. Também criou a famosa técnica conhecida como Mentalidade Mamba, na qual ele ensina que, para alcançar alto desempenho na carreira, é mais valioso o processo do que o resultado. Ou seja, para se tornar um dos melhores jogadores da NBA, Kobe precisou criar rotinas que o ajudassem a focar melhor em seus treinos.

A dedicação incansável de Kobe Bryant, refletida na Mentalidade Mamba, era focada em torná-lo o melhor no que fazia. Para isso, treinava duas vezes — ou mais — do que seus adversários, de forma inteligente e consistente. Ele não apenas praticava o que já sabia, mas também dedicava tempo extra para melhorar suas fraquezas. Como ele mesmo dizia, a matemática era simples: quanto mais você treina, melhor você fica. Essa abordagem o transformou em um dos melhores jogadores da NBA e destaca a importância do treinamento e da preparação contínua.

Para desenvolvedores de software, a Mentalidade Mamba pode ser aplicada de forma semelhante. Assim como Kobe, é essencial estabelecer rotinas e metas claras para manter o foco e a constância. A programação, assim como o basquete, é uma habilidade que pode ser aprimorada com prática constante. Quanto mais você pratica, mais natural e eficiente se torna o processo de resolver problemas e criar soluções. A chave é manter a disciplina e transformar o aprendizado em um hábito diário.

Capítulo 6
Hábito 1: Crie projetos desafiadores e construa um excelente portfólio

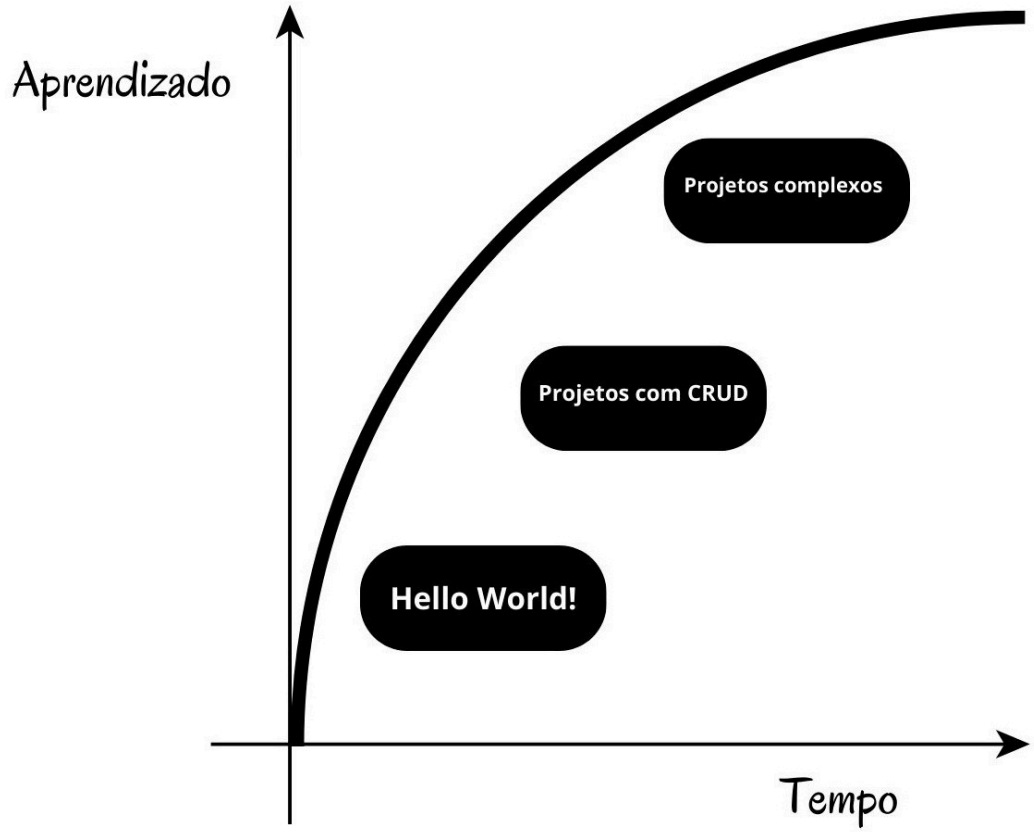

Figura 6.1: Hábito 1: sobre criar projetos desafiadores.

Desenvolver projetos de estudo aprimora significativamente suas habilidades técnicas. Frequentemente, você será desafiado a enfrentar problemas nos quais, inicialmente, não sabe nem por onde começar, ou por qual caminho seguir.

Essas situações exigirão profunda pesquisa, dedicação, tempo e energia para solucionar problemas. Além do mais, você estará trabalhando sozinho(a) e terá que lidar com erros e dificuldades que encontrará no meio do caminho. Mas esses projetos poderão contribuir para a construção de um portfólio rico e atraente.

Uma prática recomendável é aumentar gradualmente a complexidade dos projetos. Por exemplo, começar com pequenas aplicações de cadastro e busca de registros; depois, avançar para sistemas mais elaborados — como aplicações integradas a outras —, desenvolver a modelagem de bancos de dados, utilizar serviços de mensageria, aplicar conceitos de padrões de projeto e assim por diante.

Por se tratar de projetos pessoais, cujo objetivo é apenas aprender e aprimorar suas habilidades técnicas, não haverá pressão externa, demandas ou prazos para entrega. Eles podem ser desenvolvidos da maneira que você achar melhor e no seu próprio tempo. Procure divulgá-los à comunidade técnica, mostrar sua evolução, compartilhar ideias e, claro, estar aberto a sugestões e melhorias.

Quando criamos projetos pessoais e divulgamos para a comunidade, como no caso do GitHub, há vários benefícios que podemos identificar. Vejamos:

Portfólio acessível

Projetos no GitHub fornecem um portfólio de credibilidade que pode ser facilmente compartilhado. Suas ideias e seus códigos podem ajudar pessoas iniciantes ou não iniciantes e você pode receber sugestões e melhorias de outros contribuintes.

Oportunidade de colaboração

Outros desenvolvedores podem ver seu trabalho, contribuir ou até mesmo propor parcerias em novos projetos. Grandes projetos *open source* (código aberto) surgiram dessa forma, com a contribuição da comunidade técnica, e hoje são utilizados por grandes corporações ao redor do mundo.

Histórico de aprendizado

Ao revisitar seus repositórios mais antigos, você notará sua evolução e o crescimento como pessoa desenvolvedora. É sempre bom manter os projetos antigos, pois isso permite enxergar como — e onde — você evoluiu, desde a forma como escrevia o código até o modo como planejava a arquitetura.

Habituar-se a novas ferramentas

Criar projetos paralelos permitirá que você se familiarize com novas ferramentas do mercado — tecnologias que, possivelmente, não utilizaria na empresa em que trabalha, mas que terá a oportunidade de explorar no desenvolvimento dos seus próprios projetos. Entre elas, podemos citar: contêineres como Docker e Podman, orquestradores como Kubernetes, bancos de dados NoSQL, frameworks, entre outras.

Visibilidade

O GitHub é, hoje, o principal portfólio dos desenvolvedores. Ter repositórios bem-feitos e organizados na plataforma gera destaque e pode ser usado como referência por outros profissionais. Além disso, ele pode influenciar diretamente nas oportunidades de carreira. Muitas empresas utilizam o GitHub para avaliar candidatos, observando como eles estruturam seu código, quais projetos desenvolveram e em quais contribuições participaram.

Aprendizado prático e projetos pessoais

Gosto da ideia de criar projetos relacionados aos assuntos que estudo, misturando teoria e prática. Se você está estudando *Design Patterns* (Padrões de Projeto) e *Clean Architecture* (Arquitetura Limpa), após entender os conceitos teóricos, você poderá aplicá-los em seus projetos — por exemplo, na construção de APIs (*Application Programming Interface* — Interface de Programação de Aplicações), utilizando os princípios da arquitetura limpa.

Com poucas ideias claras sobre o que criar e querendo evitar complexidades desnecessárias, optei por desenvolver APIs com base em minhas séries favoritas, começando pelo *"The Office"*. A ideia era desenvolver uma API que pudesse listar, cadastrar, alterar e deletar personagens e episódios da série, sempre procurando seguir as boas práticas da Orientação a Objetos.

Disponibilizei o projeto no GitHub, permitindo que outras pessoas colaborassem ou até mesmo reutilizassem a ideia. Em pouco tempo, recebi sugestões de melhorias, estrelas e *likes* no repositório. As pessoas acharam a ideia genial e quiseram contribuir também. Segui criando vários outros projetos, mas não com o intuito de ganhar *likes* ou estrelas no GitHub, e sim com o objetivo de colocar em prática as lições aprendidas ou treinar a codificação.

Conforme você for criando mais e mais projetos, suas habilidades de programação e sua capacidade como desenvolvedor(a) tendem a melhorar. Você passa a implementar funcionalidades que, muitas vezes, no ambiente de trabalho, já estão prontas para uso — porque, no início, alguém precisou criá-las e configurá-las. Um exemplo são os módulos de autenticação ou a automatização de aplicações e serviços em contêineres. Ao trabalhar em nossos próprios projetos, não há ninguém além de nós mesmos para fazer o trabalho; entre erros e acertos, aprimoramos nossas habilidades.

Também conversei com diversos desenvolvedores para entender seus pontos de vista sobre a utilização de projetos pessoais como portfólios e ferramentas de aprendizado.

"À medida que a tecnologia avança, o acesso à informação se torna mais democrático, tornando evidente a diferença entre aqueles que se dedicam a se destacar e os que apenas se contentam com o básico. Assim, mais do que simplesmente atender a demandas, é imperativo compreender a totalidade do trabalho para incorporar experiências e conhecimentos externos a projetos realmente inovadores.

Isso eleva o impacto do desenvolvedor e enriquece o portfólio, criando oportunidades futuras. No início da carreira, projetos pessoais no portfólio são essenciais. Contudo, à medida que se ganha senioridade, o próprio portfólio se transforma, passando a abranger os projetos do dia a dia. Esses projetos não devem ser meramente entregues, mas sim pensados como cases que demonstram a visão e habilidade do desenvolvedor para usos futuros."

— Matheus Catossi, CTO na System Frame

"Se você não sabe por onde começar, não há nada de errado em recriar portfólios "clichês" encontrados em diversos tutoriais. Aliás, acredito que é um ótimo pontapé inicial, pois te trará insights sobre como iniciar o teu próprio projeto."

— Mitsuhiko Takushi, Software Engineer na Sigma Software Group

Como podemos ver, cada desenvolvedor encontra sua própria maneira de criar projetos. Alguns recriam portfólios considerados "clichês", outros fazem exercícios de cursos que adquiriram — mas todos compartilham o mesmo objetivo: aprender e aprimorar suas habilidades em programação.

Capítulo 7
Hábito 2: Aprimore sua comunicação

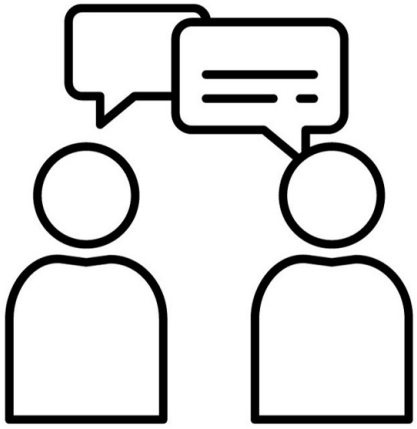

Figura 7.1: Hábito 2: aprimore a comunicação.

A comunicação é uma habilidade interpessoal que considero extremamente importante, talvez a mais crucial para qualquer carreira. No início, é comum que desenvolvedores focados em adquirir habilidades técnicas (*hard skills*) negligenciem o desenvolvimento de habilidades interpessoais — como as *soft skills*. Estas são atributos pessoais e interpessoais que potencializam a capacidade de colaborar de forma eficiente com suas equipes, de comunicar-se de forma eficaz e de saber lidar com desafios do mundo profissional.

Mesmo que você seja um desenvolvedor excepcional com habilidades técnicas robustas, uma comunicação ineficiente pode limitar seu crescimento profissional.

Historicamente, desenvolvedores de software eram vistos como profissionais mais introvertidos, frequentemente retratados em filmes de ficção científica ou em biografias como indivíduos quietos,

fechados em seu mundo de códigos. No entanto, o estereótipo do desenvolvedor solitário está mudando.

Hoje, as habilidades de comunicação são reconhecidas como essenciais não apenas para gestão pessoal, mas também para o sucesso colaborativo em ambientes tecnológicos complexos e dinâmicos — e isso é identificado desde o início de um processo seletivo. Empresas e equipes buscam profissionais técnicos que sejam bons e também tenham boa comunicação.

É importante lembrar que o profissional de TI não trabalha sozinho — ele precisa se comunicar frequentemente com sua equipe e seus gestores.

Em seu livro *"O Programador Pragmático"*, Andrew Hunt destaca a importância de uma boa comunicação em diversos níveis. *"Como desenvolvedores, temos de nos comunicar em muitos níveis. Passamos horas em reuniões, ouvindo e falando. Trabalhamos com usuários finais, tentando entender suas necessidades. Escrevemos códigos, que comunicam nossas intenções para uma máquina e documentamos o raciocínio para futuras gerações de desenvolvedores. Redigimos propostas e memorandos para solicitar e justificar recursos, além de relatar abordagens. E trabalhamos diariamente dentro de nossas equipes para defender nossas ideias, modificar práticas existentes e sugerir práticas novas. Uma grande parte do nosso dia é gasta em comunicação, portanto, temos de fazer isso bem."* (O Programador Pragmático: De Aprendiz a Mestre, 2010, p.40)

Vejamos algumas razões pelas quais a comunicação é tão importante no desenvolvimento de software:

Entendimento de requisitos

O papel do desenvolvedor não é escrever código com aquela linguagem que tem aquela sintaxe simpática e amigável, ou o último (*framework*) do mercado, não. Como já havia dito nos capítulos

anteriores, o papel da pessoa desenvolvedora é resolver problemas de negócio usando a tecnologia como meio.

Desenvolvedores precisam se comunicar efetivamente com as equipes de negócios para entender suas necessidades e converter isso em requisitos técnicos. Uma comunicação ideal ajuda a entender os requisitos de negócio e alinhar as expectativas dos *stakeholders*.

Resolução de problemas

A falta de uma boa comunicação leva a mal-entendidos sobre requisitos do projeto, criação e desenvolvimento das tarefas e resolução de problemas. Por exemplo, imagine um projeto onde o(a) desenvolvedor(a) não entendeu completamente um requisito de negócio por falta de comunicação clara. Esse mal-entendimento pode levar ao desenvolvimento de uma funcionalidade que não atinge as necessidades do usuário ou cliente.

Em outra situação, podemos ver o desafio de uma equipe na integração de um determinado sistema, e por meio de reuniões de alinhamento utilizando diagramas claros e uma comunicação eficiente, a equipe conseguiu entender e resolver o problema mais rapidamente.

Transmissão de conhecimento

Compartilhar conhecimento com outros membros do time, seja através de apresentações, *code review*, *pair programming* ou documentações técnicas, exige a habilidade de expressar ideias de forma clara e objetiva.

Prevenção de conflitos

Muitos conflitos em projetos de tecnologia surgem de falhas na comunicação. Habilidades comunicativas aprimoradas podem evitar

esses conflitos, facilitando a transmissão de informações claras e promovendo melhor entendimento.

Negociação e persuasão

Argumentar efetivamente é essencial, seja para convencer a empresa ou o gestor a adotar uma nova tecnologia ou negociar prazos com o time de negócio. Uma comunicação assertiva é a chave para persuasão eficaz.

Comunicar-se bem em processos seletivos

Uma boa conversa transparente e objetiva sobre suas ideias e experiências, bem como a habilidade de responder a perguntas de forma direta, são bem pontuadas durante a entrevista de emprego. A primeira impressão é a que fica, e uma boa comunicação neste contexto pode trazer muitos ganhos.

Experiências práticas

Durante minha passagem na Globo, trabalhei com o gestor Vitor Cardoso, que considero uma das pessoas mais influentes e com grande domínio na arte de se comunicar bem. Aprendi muito com o Vitor sobre a importância da comunicação e, principalmente, pude evitar diversos conflitos usando a comunicação de forma eficaz. Por isso, não pude deixá-lo de fora deste capítulo. Perguntei a ele sua visão sobre a comunicação ineficiente nos times de tecnologia e qual o impacto disso.

"Quantas vezes você já ouviu a frase: 'Isso foi problema de comunicação'? Isso ocorre em todas as profissões, mas algumas geram mais impacto do que em outras. Quando estamos trabalhando em times de desenvolvimento, esse é o tipo de impacto em cascata. Poucas pessoas procuram desenvolver essa habilidade, sempre acham quem ao falar e escrever já estão treinando a sua comunicação, mas esse é um ponto que precisa ser avaliado constantemente para que, de fato, tenha uma evolução.

Quanto maior o time, mais complexo se torna esse canal, não porque é apenas mais um, mas sim porque o crescimento é exponencial. O comentário de uma pessoa pode desfazer toda a comunicação gerada por outra pessoa, e isso é uma atividade complexa.

Um entendimento diferente, uma dúvida não tirada, um complemento não realizado ou qualquer contribuição guardada ou não discutida pode ser o suficiente para a não compreensão ou para um retrabalho coletivo. Essa diferença tem que ser buscada e deve-se analisar constantemente a forma de se comunicar. Não ache que é uma atividade simples, requer dedicação diária para a evolução."

— Vitor Cardoso, Líder de Equipe na Globo

Na visão de Mitsuhiko, que é desenvolvedor, a comunicação ineficiente compromete o crescimento profissional, especialmente em situações que exigem pedir ajuda ou esclarecer dúvidas sobre o que precisa ser feito.

"Se algo não ficou claro para você, manifeste-se. Caso contrário, os seus colegas acreditarão que você entendeu tudo. Não entendeu tudo? Pergunte. Precisa de ajuda? Peça. Isso é importante para os seus pares, assim, eles entenderão em que estágio sua carreira está e vão poder guiar você de acordo. Não deixe o ego atrapalhar o seu crescimento."

— Mitsuhiko Takushi, Software Engineer na Sigma Software Group

Existem algumas estratégias que você pode aplicar para aprimorar sua comunicação, vejamos:

Treinamento formal

Participar de workshops, palestras ou cursos de comunicação eficaz é uma excelente maneira de aprimorar suas habilidades de comunicação. Por exemplo, plataformas como LinkedIn Learning e

Coursera oferecem uma variedade de cursos, desde comunicação interpessoal até comunicação avançada focada em apresentações.

Há também livros como: *A Arte de se Comunicar* (Thich Nhat Hanh, 2017), *Como Fazer Amigos e Influenciar Pessoas* (Dale Carnegie, 2019), *O Corpo Fala* (Pierre Weil, 2015) etc. Cursos e livros podem ajudar a melhorar a forma de se expressar, a comunicação com a fala e o corpo, reconhecer os momentos de escutar e de falar, vícios de linguagem etc.

Prática regular

Uma boa maneira de praticar no trabalho é fazer pequenas apresentações em grupo, conduzir a *daily* ou outras cerimônias com o time de desenvolvimento.

Você pode criar um blog e escrever sobre sua rotina ou sobre temas que despertam seu interesse, como forma de aprimorar sua comunicação escrita.

Feedbacks construtivos

Sugira à sua liderança técnica ou à sua gestão que lhe repassem comentários sobre seu trabalho. Isso contribui para melhorar pontos importantes e gerar discussões construtivas.

Em resumo, a comunicação atua como uma ponte entre a complexidade técnica e as necessidades do negócio, sendo fundamental para o sucesso na carreira de desenvolvimento de sistemas e em qualquer outra área. Esse é um ponto que abordei no capítulo sobre metodologias ágeis, onde a comunicação é utilizada constantemente entre o time de desenvolvimento e o time de negócio.

Hoje, você pode estar focado(a) em escrever bons códigos. No futuro, poderá ser o(a) profissional que transforma uma ideia inovadora em um negócio próspero, e precisará apresentá-la a investidores, para os quais pode não fazer diferença como os códigos são escritos nem quais ferramentas são utilizadas.

CAPÍTULO 8
Hábito 3: Organize seus estudos e foque no essencial

Figura 8.1: Hábito 3 organize seus estudos.

A organização é essencial para quem mantém uma rotina de estudos. É fundamental saber o que e como estudar de maneira eficaz para reter melhor o conhecimento.

Um estudo desorganizado e sem estratégia pode ter impactos negativos: pode resultar em um grande desperdício de tempo e em menor retenção de informações, criando ciclos de estudo ineficientes e frustrantes.

Neste capítulo, vou explorar o tema do processo de aprendizagem, compartilhando técnicas para otimizar a absorção de conhecimento. Ao compreender melhor como nosso cérebro funciona durante a aprendizagem, podemos organizar e focar nossos estudos com eficácia e eficiência. De nada adianta acumular cursos e ler centenas de livros se não conseguir efetivamente assimilar as informações.

Uma referência valiosa que compartilho neste capítulo é o livro *"Já Entendi"*, de Gladys Mariotto. Nesta obra, Gladys apresenta técnicas para uma melhor absorção de conhecimento e detalha como nosso cérebro processa as informações durante o estudo. O livro proporciona uma visão ampla sobre as estratégias cerebrais de retenção de conhecimento, e descreve três processos que ocorrem durante a aprendizagem, que são:

Ler

A leitura proporciona uma aproximação inicial com o conteúdo, que pode ser apresentado em forma de textos, cálculos, exercícios ou resolução de problemas.

Organizar

Essa fase é fundamental para definir melhores formas de estruturar suas leituras, que podem incluir resumos coloridos, fichamentos, mapas mentais ou uma combinação destes métodos. Quanto mais sentidos forem estimulados durante essa etapa, melhor será a retenção.

Assimilar

Este é o momento em que a memória trabalha para criar e reter o conteúdo, sendo a chave para a aprendizagem. No livro, Gladys menciona que o conhecimento prévio ajuda no processo de retenção das informações. O conhecimento prévio refere-se àquilo que já sabemos e serve como alicerce para a aquisição de novos saberes.

Por exemplo, se você está aprendendo uma nova linguagem de programação ou explorando um conceito tecnológico e já tem familiaridade com o assunto, seja por meio de leitura ou ouvindo podcasts, isso constitui o seu conhecimento prévio. Ter essa base

facilita e acelera o processo da aprendizagem do novo conteúdo que você está estudando.

Pirâmide da aprendizagem

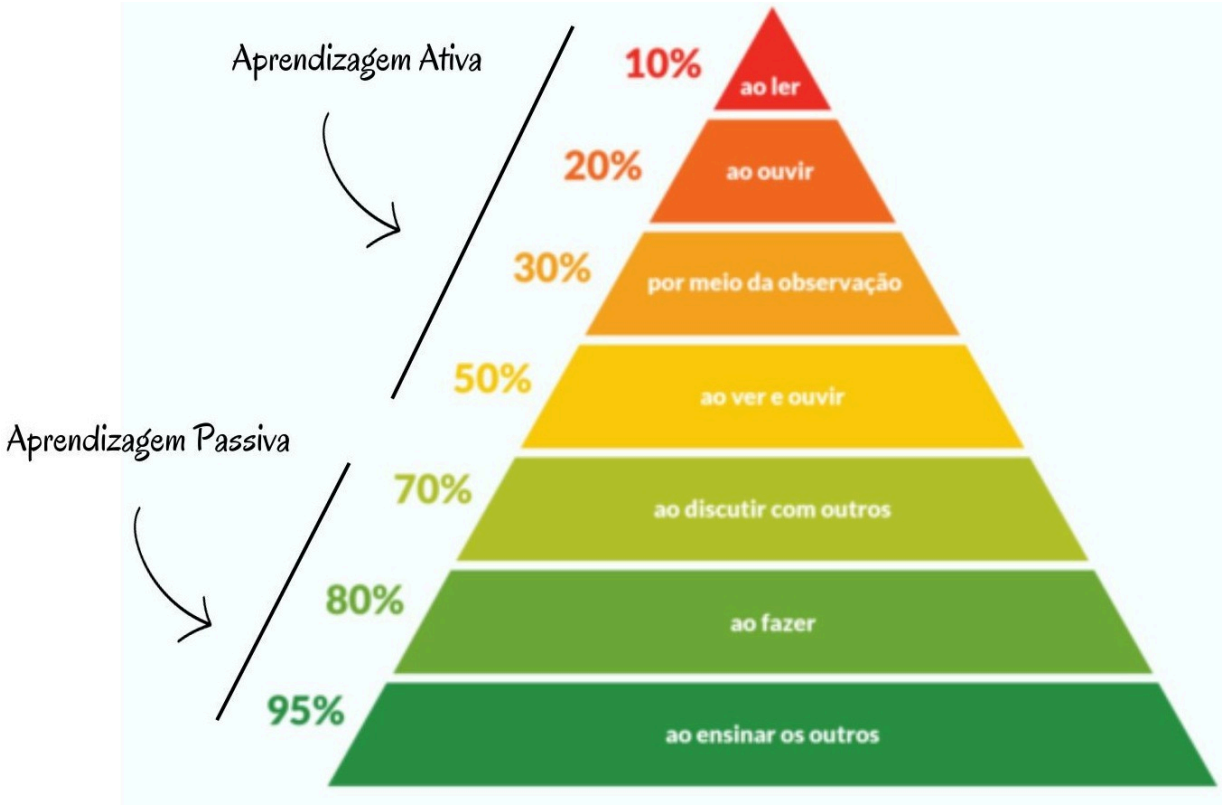

Figura 8.2: Hábitos de estudo.

O psiquiatra William Glasser (1925-2013) foi um cientista e pesquisador norte-americano e idealizador da pirâmide da aprendizagem, um método de aprendizado de maneira esquemática. Nele, podemos examinar como o cérebro humano armazena as informações.

Esse modelo sugere os seguintes percentuais de retenção de conhecimento:

- 10% do que lemos;
- 20% do que ouvimos;
- 30% por meio da nossa observação;

- 50% por meio de ver e escutar;
- 70% quando discutimos ideias com outros;
- 80% do que fazemos;
- 95% do que ensinamos aos outros.

Como podemos observar, na representação teórica da pirâmide, retemos a maior porcentagem de conhecimento quando ensinamos. Ou seja, a melhor forma de aprender e reter o conhecimento é transferindo-o. Por exemplo, ao ensinar um amigo sobre um novo conceito de programação, você não apenas revisa o conteúdo, mas também identifica lacunas em seu próprio conhecimento.

Essa estratégia de ensinar pode ser aplicada de forma prática, como criar tutoriais ou artigos técnicos sobre algum tema que você esteja estudando e queira fixar mais profundamente. Já dizia Cora Coralina (1889-1985), *"feliz é aquele que transfere o que sabe e aprende o que ensina"*.

Entretanto, é importante salientar que a autenticidade e a precisão dos percentuais indicados pela pirâmide da aprendizagem são frequentemente questionadas pela comunidade científica. Não há evidências empíricas robustas que sustentem esses números exatos, e especialistas em educação sugerem que as taxas de retenção podem variar significativamente de acordo com o indivíduo e o contexto. Portanto, esse modelo deve ser visto como uma representação teórica e não como uma garantia dos resultados do aprendizado.

Como organizar seus estudos e reter melhor o conhecimento

Agora que você compreende o funcionamento da aprendizagem e como reter melhor o conhecimento, é hora de colocar isso em prática. A seguir, apresento algumas técnicas e estratégias para aprimorar sua organização nos estudos:

- Técnica de Pomodoro: divida seu tempo em blocos de 25 minutos de foco total, seguidos por 5 minutos de pausa. Após quatro ciclos, faça uma pausa mais longa. Use os blocos de foco para estudar teoria e aplicar a prática. Durante as pausas de cinco minutos, beba água, caminhe um pouco ou simplesmente relaxe. Esse momento serve para a descompressão.

- Anotações estruturadas: durante a leitura ou o estudo, faça anotações claras e organizadas. Recomenda-se a escrita à mão em cadernos, blocos de anotações ou até mesmo em post-its. Claro, você também pode optar por ferramentas digitais, como Notion, Evernote, Google Docs, entre outras. Crie mapas mentais ou diagramas para facilitar a revisão e mantenha-os sempre visíveis em sua mesa de trabalho ou no local onde estiver estudando.

- Agenda de estudos: identifique e separe os tópicos mais importantes para seu conhecimento e priorize-os. Crie uma agenda de estudos para determinar tempo suficiente para estudar cada tópico ou matéria. Por exemplo, "toda segunda, às sete horas da manhã, estudar Estrutura de Dados durante duas horas" — e assim por diante, até que isso se torne um hábito.

- Conhecimento prévio: antes de mergulhar em um novo assunto, revise conceitos relacionados que você já domina. Isso cria uma base sólida para o novo aprendizado. Por exemplo, antes de estudar microsserviços, revise os conceitos de APIs e Arquitetura de Software. Em seguida, faça testes e avaliações para verificar esses conhecimentos. Caso não se saia bem nos resultados, revise novamente.

- Estudo ativo: em vez de apenas ler ou assistir a vídeos, pratique o estudo ativo. Isso inclui fazer perguntas a si mesmo, resolver problemas, ensinar o conteúdo a outra pessoa ou até mesmo gravar-se explicando o assunto. Quanto mais ativo for o seu envolvimento com o material, melhor será a retenção.

Técnica Rubber Duck Debugging

Quando estamos focados em solucionar um determinado problema ou depurando uma aplicação para encontrar um bug, nossa mente se exaure e acaba deixando passar pequenos detalhes. Nesses momentos, é sempre bom depurar a aplicação em conjunto — afinal, duas cabeças pensam melhor do que uma. Se uma falha, a outra está ali como alicerce para enxergar o que a primeira não percebeu. Comunicar-se e testar hipóteses em conjunto também é uma excelente técnica.

Gosto de utilizar uma técnica bastante conhecida chamada *Rubber Duck Debugging* (depuração do pato de borracha). Trata-se de um método de depuração de código em que o problema é articulado em linguagem natural ou escrita, utilizando um pato de borracha como interlocutor simbólico. O nome faz referência à história do livro *The Pragmatic Programmer* (O Programador Pragmático), em que um programador depurava o código forçando-se a explicá-lo linha por linha a um pato de borracha. Existem outros termos e variações dessa técnica, muitas vezes envolvendo diferentes objetos ou até mesmo animais de estimação, como um cachorro ou um gato.

Essa é uma excelente técnica para fixar o aprendizado. Quando você explica o conceito em voz alta (mesmo para um objeto), seu cérebro organiza informações de forma lógica, identifica lacunas no seu entendimento ("eu sei fazer, mas não sei explicar o porquê") e força conexões entre a teoria e a prática.

Alguns desenvolvedores que entrevistei compartilharam a maneira como organizam seus estudos e como conciliam o tempo de estudo com a jornada de trabalho.

"Quando busco aprimorar minhas habilidades, costumo escolher tópicos que ainda não domino e me dedico a eles nos finais de semana, durante meus estudos. Minha estratégia de aprendizado envolve a leitura de artigos em plataformas como o Medium, Dev.to e Quora, além da consulta de documentações quando disponíveis.

Também busco cursos e tutoriais, e aplico o conhecimento em meus projetos pessoais.”

— Ruan Campelo, Software Engineer na Globo

“Se tudo for importante, nada é importante. Priorize estudar e focar em um único assunto ou linguagem por 2 a 3 meses ou conforme necessidade. Isso permite que sua mente esteja completamente imersa no tema, e, consequentemente, acelera a sua evolução.”

— Mitsuhiko Takushi, Software Engineer na Sigma Software Group

“Costumo organizar tópicos e subtópicos, atribuindo uma ordem de prioridade. Essa prioridade depende em boa parte do projeto que estou atuando no trabalho ou em algum projeto pessoal; no entanto, busco dar foco em três principais temas, que são:

1. *Desenvolvimento: assuntos como programação, estrutura de dados, linguagem de programação, frameworks e afins.*
2. *Arquitetura de Software: com foco voltado para padrões de projetos, system design, cloud computing etc.*
3. *Soft Skills: assuntos para melhorar habilidades como comunicação, trabalho em equipe e liderança. Considero algo muito importante, e assim que assumi o papel de Tech Lead, passou a ser um dos meus principais focos de estudo.*

Ao estudar um determinado tema de desenvolvimento, por exemplo, um novo framework ou uma linguagem de programação, inicialmente procuro entender os conceitos básicos por meio de artigos, livros, vídeos, palestras e podcasts. Após esse entendimento, tento desenvolver alguma POC ou um projeto bem simples apenas para consolidar o entendimento dessa tecnologia.

No caso de estar estudando uma nova linguagem de programação, uma prática que recomendo é reescrever algum projeto ou aplicação que você já tenha feito em sua linguagem habitual. Isso sempre me ajudou a me familiarizar mais rapidamente com a linguagem de programação que estou estudando.

Por fim, recentemente, comecei a estudar desenvolvimento de jogos eletrônicos. Embora isso não seja algo relacionado ao meu trabalho atual, foi uma forma que encontrei para buscar novos desafios. Até o momento, tem sido algo totalmente diferente do que eu já havia estudado. Isso vem me ajudando a aprimorar minhas habilidades como engenheiro de software.”

— Reginaldo Barros, Tech Lead na Platform Builders

"Eu costumo ler livros, artigos e fazer muitas provas de conceito, tanto no trabalho quanto fora dele. Testar rápido uma ideia é essencial tanto para projetos grandes quanto para resolver desafios de LeetCode. Esse feedback faz a pessoa iterar de forma mais rápida na ideia, descartá-la e evoluí-la de forma saudável."

— Maurício Antunes, Software Senior Engineer na Clerk.com

“A minha rotina de estudos na verdade é composta de vários experimentos para entender o que funciona para mim no momento atual. Nesses experimentos, passei a entender que os rituais e o ambiente são importantes para preparar o meu cérebro para o momento de foco. Por exemplo, eu trabalho no meu escritório, mas gosto de estudar em outro cômodo da casa, como a sala.

Eu já passo muitas horas no meu escritório e essa troca de ambiente ajuda o meu cérebro a entender que vou fazer algum tipo de atividade fora do contexto de trabalho. Para priorizar meus estudos, uso dois critérios: o que é essencial para o meu trabalho e quais temas despertam minha curiosidade.

Hoje eu entendo que só consigo fazer essa priorização depois de algum tempo de experiência trabalhando na área. Quando estava começando, ficava desesperada com tanta coisa para estudar. Hoje consigo definir com clareza quais serão os meus próximos passos, fazer um planejamento da minha rotina e aceitar que sempre vai existir algo que eu ainda não sei.

As regras que sigo para meus estudos são:

1. *Dedicar pelo menos três dias por semana entre uma e duas horas por dia, após o trabalho;*
2. *Quando estudo teoria, como livros ou documentação, faço anotações, releio e realizo pesquisas adicionais;*
3. *Praticar resolvendo exercícios ou desenvolvendo projetos para aplicar o que foi aprendido teoricamente;*
4. *Criar conteúdo sobre o aprendizado. Isso pode ser um README do projeto, artigos, vídeos ou posts em redes sociais. Se tenho dificuldades nesta etapa, volto ao estudo teórico para consolidar a compreensão. Esse conteúdo, além de me ajudar, reforça minha autoridade no assunto e pode auxiliar outras pessoas. Ao compartilhar publicamente o código no GitHub, artigos ou conteúdos em redes sociais, posso receber feedbacks que oferecem perspectivas diversas, novas ferramentas ou métodos diferentes relacionados ao que criei.*

Minha rotina de estudos é adaptável. Às vezes a inspiração vem e fico estudando por vários dias consecutivos. Porém, nunca passo de duas horas na atividade, estabeleço limites diários a mim mesma para ter um balanço de vida social, estudos e trabalho mais saudáveis. Então, quando chega a hora de parar, eu paro.

Outro ponto legal para falar sobre isso é que não estudo apenas desenvolvimento de software. Além de tecnologia, gosto de ler sobre desenvolvimento pessoal, ciência, finanças, política e história. E faço com a leitura um estudo ativo também, destaco e escrevo notas que considero importantes. Por isso gosto de dedicar apenas três dias para estudos de desenvolvimento. No restante do tempo, costumo fazer alguma atividade diferente ou descanso. E aqui vai outra coisa que aprendi depois de dois burnouts: o descanso também faz parte da rotina de estudos e deve ser incluído nela."

— Laís Lima, Software Engineer na Avenue e Criadora de Conteúdo

Organizar seus estudos e focar no essencial não é apenas uma questão de produtividade, mas também de eficiência do aprendizado. Como vimos no decorrer deste capítulo, técnicas como Pomodoro, anotações estruturadas, testes e revisões podem reter melhor o conhecimento do assunto.

Além disso, procure criar ambientes de estudo adequados, defina metas claras e realistas para alcançar seus objetivos. Vimos ainda outra estratégia, baseada na pirâmide da aprendizagem, que é ensinar outras pessoas, discutir em grupo ou até escrever sobre o tema. Explicar conceitos com suas próprias palavras ajuda a consolidar o aprendizado.

Capítulo 9
Hábito 4: Domine o inglês, a língua da tecnologia

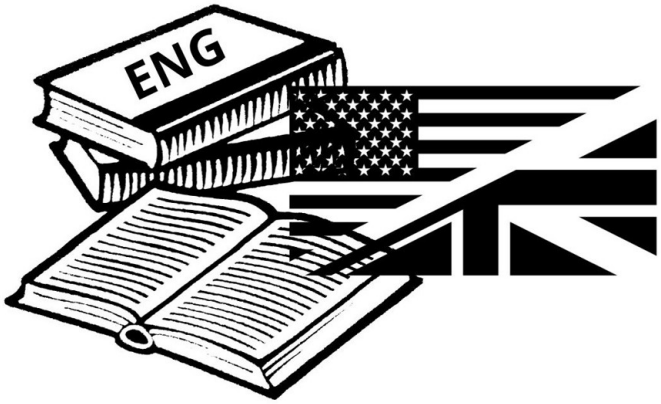

Figura 9.1: Hábito 4: domine o inglês.

Um estudo publicado em 2024, de autoria de Fabricio Silvestre, Lucas Oliveira e Ruan Moura, destaca que o inglês se consolidou como idioma predominante da área de tecnologia da informação. Segundo os autores, essa influência vem desde o surgimento das *Big Techs* e se mantém até hoje devido ao uso do inglês na criação e evolução de softwares, ferramentas, componentes e *hardware*. Sem contar que grande parte da documentação técnica, linguagens de programação e fóruns de discussão são escritos nesse idioma, tornando-o essencial para profissionais que desejam se aprofundar nesta área. (Faculdade de Tecnologia de Jundiaí, 2024. Acesse a pesquisa em https://ric.cps.sp.gov.br/handle/123456789/21438).

Essa predominância do inglês também reflete no mercado de trabalho. De acordo com a pesquisa da *Global Worker 2024*, realizada pela Husky, 58,9% dos brasileiros que trabalham para o exterior atuam no setor de tecnologia da informação.

Portanto, esses dados reforçam a prevalência do idioma não apenas para quem deseja consumir conteúdo técnico, como também para aqueles que buscam oportunidades de carreira internacional. (Husky, 2024. https://www.husky.io/global-worker-2024).

O inglês permite que desenvolvedores se engajem em comunidades globais, fóruns e eventos internacionais, como conferências e webinars. Nessas comunidades, é possível compartilhar conhecimento, promover produtos, criar conexões profissionais, esclarecer dúvidas e fazer *networking*.

Vale destacar também a oportunidade de participar de projetos (*open source*) e colaborar em desafios tecnológicos com profissionais de todo o mundo.

Já entrevistei diversos profissionais em meu podcast que tiveram a oportunidade de fazer carreira fora do país. O que abriu a porta foi o perfil colaborativo para projetos *open source*, ou até mesmo a publicação de conteúdos técnicos, em inglês.

A verdade é que, mesmo com um perfil técnico excelente, sem o inglês eles não teriam ido tão longe. Essa habilidade linguística é importante também para compreender documentações técnicas, o que facilita a solução de problemas.

O(a) profissional que possui o domínio do inglês tem a possibilidade de fazer uma carreira internacional e muitas vezes trabalhar nas mais desejadas *Big techs*, gigantes da tecnologia. Hoje em dia há um número significativo de brasileiros que trabalham remotamente, a partir do Brasil, para empresas do exterior. Essa foi uma das motivações que observei ao conversar com outros profissionais que decidiram investir tempo e dinheiro para melhorar suas habilidades no idioma e fazer um *upgrade* de carreira.

Uma matéria do InfoMoney destaca o aumento expressivo de trabalhadores residentes no Brasil que estão empregados por empresas estrangeiras, o que representa um crescimento de 491%. Esses dados são provenientes de um estudo realizado pela fintech

brasileira Husky, uma empresa de tecnologia de finanças e atendimento pessoal.

Segundo a matéria do InfoMoney, "conforme a pesquisa, em março de 2020, no início da pandemia, havia 1.251 usuários da Husky. No fim de novembro de 2022, eram 11.284. O total de profissionais que trabalham do Brasil para empresas do exterior pode ser maior, pois nem todos usam a plataforma que fez a pesquisa." (InfoMoney, 2023. Acesse os dados em https://www.infomoney.com.br/carreira/numero-de-profissionais-que-trabalham-para-o-exterior-cresceu-491-veja-dicas-de-como-fazer-o-mesmo/).

Outra matéria, publicada pela revista Exame, apontou que mais brasileiros estão buscando trabalhar remotamente para empresas estrangeiras. Houve um aumento expressivo de 70% para 78% comparado com os últimos anos. Segundo a matéria, os principais motivos que atraem os profissionais são o recebimento do salário em uma moeda mais forte e o trabalho remoto. “A possibilidade de uma melhor remuneração em comparação aos padrões locais, incluindo o ganho em moedas fortes como o dólar, libra e euro, certamente é um grande atrativo. Atuar sem sair de casa também é outro grande atrativo, especialmente quando boa parte das empresas brasileiras retomaram o formato presencial”, diz Marcelo Gripa, cofundador e diretor de operações da Futuros Possíveis. “Conforme aponta a nossa pesquisa, o interesse de trabalhar fora do Brasil nesse contexto aumentou na comparação com 2013 (de 70% para 78%), um indicativo de que a volta ao trabalho presencial é questionada.” (Exame, 2024. https://exame.com/carreira/nova-fuga-de-cerebros-brasileiros-buscam-agora-trabalhar-de-home-office-para-empresa-internacional-2/).

Mais uma pesquisa, realizada e publicada pela fintech Husky em 2024, mostra que a Engenharia de Software é a área com maior número de profissionais nas empresas, com 51,6%. Entretanto, o percentual caiu 11,9% em relação ao ano anterior (63,5%), o que significa que outras áreas podem ter ganhado mais espaço, como o

Marketing, Financeiro e Recursos Humanos. Os dados apontam que 96,7% dos entrevistados têm pelo menos um segundo idioma além do português, sendo a maioria o inglês, e depois o espanhol, presente em 34,7% dos currículos. (Husky, 2024. https://www.husky.io/global-worker-2024).

Muitos brasileiros enfrentam barreiras para alcançar a fluência no idioma devido a fatores culturais e socioeconômicos. Embora existam iniciativas que buscam tornar o aprendizado mais acessível, essas ainda não são suficientes para eliminar todas as dificuldades.

Em regiões periféricas, por exemplo, apesar do aumento no acesso a conteúdos educativos gratuitos online, a falta de aplicação prática imediata do inglês pode desmotivar seu aprendizado. Além disso, o ensino do inglês nas escolas públicas, muitas vezes, é básico demais e insuficiente para uma proficiência completa. Isso evidencia a necessidade de reformas educacionais e de iniciativas que valorizem a importância deste idioma, tornando seu aprendizado mais acessível e relevante para todos.

De todo modo, com a expansão do acesso a materiais educativos na internet, aprender inglês ou aprimorar a fluência no idioma tornou-se mais acessível do que nunca. Atualmente, existem centenas de aplicativos de línguas, canais no YouTube que ensinam inglês a partir do básico, pessoas que contribuem com conteúdos educativos em redes sociais e sites com exercícios para fixação.

Para muitos, a falta de oportunidade para aplicar o conhecimento que foi aprendido, seja por meio de escrita regular ou de conversação com algum nativo, pode desmotivar o aprendizado. Em áreas centrais de algumas capitais, há grupos de conversação que têm como objetivo praticar o idioma com turistas hospedados. Embora essas atividades sejam gratuitas e ocorram em locais como bares, cafés ou hostels, elas podem ser desafiadoras para quem mora em regiões periféricas ou mais afastadas do centro em virtude da distância e do tempo necessário para o deslocamento.

Todavia, existem aplicativos voltados para a prática da conversação com outras pessoas, sejam elas do mesmo nível ou mais avançadas, estrangeiras ou até mesmo brasileiras. Por exemplo, no Discord há canais de inglês com foco em conversação; o HelloTalk permite conversar com falantes nativos de inglês. Já o SpeakingPal oferece lições envolventes que incentivam os usuários a desenvolverem suas habilidades. E, por fim, temos a inteligência artificial, como o ChatGPT, que permite, por meio da sua funcionalidade de voz, discutir qualquer assunto em qualquer idioma.

Algumas mudanças de atitude e hábito, tanto gratuitas como pagas, podem beneficiar seu aprendizado no inglês. Vamos ver algumas.

Estude com um professor particular

Estudar com um professor particular pode ser muito benéfico. O professor pode corrigir seus erros e orientá-lo tanto com relação à pronúncia e à gramática, evitando o desenvolvimento de pequenos vícios de linguagem. Além disso, as aulas podem ser personalizadas de acordo com suas necessidades, como a escrita de texto, a preparação para apresentações, a prática de diálogos, entre outras.

Pratique a escrita

Pratique a escrita regularmente. Comece escrevendo pequenos textos sobre sua rotina diária e gradualmente avance para redações sobre temas que você domina. Compartilhe esses textos com seu professor para que ele possa revisar e corrigir. Tornar isso um hábito em suas aulas ajuda a melhorar significativamente seu vocabulário e gramática.

Consuma conteúdos em inglês

Consuma conteúdos em inglês, seja através de vídeos, artigos, livros técnicos ou documentações de ferramentas. É uma excelente forma

de aprimorar o idioma.

Uma boa estratégia é assistir a filmes ou séries em inglês com legendas em português, progredindo para legendas em inglês e, eventualmente, assistindo sem legendas. Aos poucos, você perceberá que a compreensão ficará mais natural.

Aplicativos e grupos de línguas

Os aplicativos de línguas são uma ótima opção para quem deseja aprender um idioma, especialmente o básico. Duolingo, Babbel e TalkPal são ferramentas excelentes para iniciantes. Além disso, existem comunidades no Discord nas quais é possível praticar a conversação por videoconferência em grupo.

Estratégias para alcançar uma oportunidade de trabalho internacional

Conversei com a Alessandra Souza, que hoje mora e trabalha em Toronto, Canadá. Ela compartilhou as estratégias que utilizou para conseguir essa oportunidade de trabalho internacional e como foi a experiência durante o processo seletivo.

"Desde o momento em que decidi buscar oportunidades de emprego internacional, meu objetivo não se limitava apenas a ganhar em outra moeda, mas também a adquirir experiência e imersão em diferentes culturas. Para atingir esse objetivo, elenquei os requisitos exigidos pelas vagas internacionais e identifiquei áreas em que precisava aprimorar meus conhecimentos, seja por meio de estudos ou avanço na minha carreira.

Inicialmente, desejava trabalhar com desenvolvimento, mas estava inserida em uma equipe de arquitetura, distante da programação. Diante disso, optei por mudar de emprego e ingressar em uma empresa que utilizava as tecnologias que almejava agregar ao meu currículo. Um ano após essa transição, ao me dedicar integralmente à busca por oportunidades internacionais, me senti confiante com

meu conhecimento das tecnologias em alta. No entanto, enfrentei dificuldades com o idioma inglês e até fui reprovada na primeira etapa de um processo seletivo devido à falta de fluência.

Em outra ocasião, recusei uma proposta de emprego em uma empresa onde o inglês seria necessário apenas em algumas reuniões, algo que não se alinhava com meus objetivos naquele momento. Na fase final de entrevistas para minha posição atual, deparei-me com desafios para compreender alguns termos técnicos em inglês devido ao sotaque do entrevistador. Precisei solicitar que ele repetisse diversas frases e, em certa ocasião, pedi para que escrevesse os termos no chat do Teams para garantir o entendimento correto. Naquele momento eu tinha certeza de que seria reprovada.

Meses após integrar a equipe da empresa, questionei um colega sobre o motivo de terem me escolhido, considerando as dificuldades com o inglês. A resposta dele ressaltou que minha determinação e persistência em tentar diversas formas de comunicação, inclusive solicitando informações por escrito, evidenciaram uma qualidade valorizada por eles: a busca por compreender completamente antes de responder ou começar a programar.

Compreendi, ao longo do tempo, que a fluência em inglês é crucial, mas também percebi a importância da comunicação em geral, da capacidade de me expressar claramente e da habilidade em formular perguntas. Minha prioridade é garantir que meu interlocutor compreenda efetivamente o que desejo transmitir, razão pela qual concentro meus esforços em ser clara, objetiva e eficaz em minha comunicação."

— Alessandra Souza, Software Engineer na Environics Analytics

Aprender inglês é um processo contínuo que envolve prática constante e exposição ao idioma. Independentemente do método escolhido, seja estudando com um professor, praticando a escrita, consumindo conteúdos em inglês ou participando de grupos de

conversação, o mais importante é manter a consistência. Quanto mais você se desafiar e se expor ao idioma, mais natural será sua evolução.

Busque ler mais artigos técnicos, assista a vídeos em inglês que expliquem conceitos sobre Engenharia de Software. Isso vai facilitar o entendimento em relação a termos técnicos com os quais você lida no dia a dia.

CAPÍTULO 10
Hábito 5: Participe e contribua com a comunidade Tech

Figura 10.1: Hábito 5: participe e contribua com a comunidade tech.

Quando iniciei minha jornada na área de tecnologia, a internet não disponibilizava tanto conteúdo técnico quanto hoje. Durante os anos de faculdade, raramente tínhamos eventos e palestras, e, quando tínhamos, não eram financeiramente acessíveis para muitos alunos. A busca por conhecimento era mais desafiadora, e a maior parte do meu aprendizado vinha de livros emprestados da biblioteca da faculdade. Além dos livros, alguns fóruns da época foram essenciais para minha prática em exercícios de programação: destaco aqui nomes como Gustavo Guanabara, Maujor (Dinossauro do CSS), José Macoratti e, claro, StackOverflow. Foi através desses websites e fóruns que pude fixar meu aprendizado fora da faculdade.

Essa realidade começou a mudar significativamente após minha formação, já em meados de 2014, quando descobri plataformas que facilitavam a interação e o aprendizado contínuo.

Por exemplo, o aplicativo *Meetup* (https://www.meetup.com/pt-BR/) é uma plataforma cujo propósito é unir pessoas com interesses em comum, facilitando a criação de grupos e organização de eventos. Funciona da seguinte forma: você cria uma conta no *Meetup*, escolhe tópicos de interesse, como tecnologia, dança, aventura, entre outros, e ele sugere grupos relacionados aos seus interesses. Os organizadores dos grupos, então, planejam eventos para promover o encontro e a interação entre os membros. E, naquela época, foi lá que conheci muitas comunidades de tecnologia, nas quais fiz parte de eventos, fiz *networking* e estive em *hackathons*.

O *hackathon* é um evento de programação em que equipes multidisciplinares desenvolvem soluções tecnológicas para um desafio específico, e, na maioria das vezes, as empresas que patrocinam esses eventos concedem premiações aos participantes, incluindo recompensas em dinheiro ou bolsas de estudo. Minha primeira participação em um *hackathon* foi em 2016. Tive uma experiência bem marcante — fiz muitos amigos da comunidade Tech, aprendi novas tecnologias e pude enxergar como resolver grandes problemas com o uso da tecnologia, sempre com uma visão de negócio.

A comunidade Tech é incrivelmente colaborativa. Ela não apenas compartilha conhecimento e reúne pessoas com interesses em comum, mas também tem o poder de transformar grandes projetos em realidade, contando apenas com o apoio mútuo de seus membros.

Muitos projetos *open source* surgiram com a ajuda da comunidade Tech ou foram inicialmente criados por grandes empresas. Hoje, são mantidos por colaboradores da comunidade global. Exemplos incluem o React, um *framework* de JavaScript que foi criado em 2011 pela Meta para uso interno e, posteriormente, apresentado como projeto *open source*; o Git, um sistema de controle de versionamento de código, desenvolvido em 2005 por Linus Torvalds, o criador do Linux; e o Apache, um dos servidores web mais

populares, originado e mantido por uma comunidade aberta de desenvolvedores.

Não posso deixar de comentar também sobre a rede social BlueSky, uma plataforma de microblogues descentralizada construída com base no AT Protocol, um protocolo de código aberto que permite a interoperação entre aplicativos sociais. Essas são apenas algumas ferramentas e aplicativos *open source*.

Como podemos ver, a comunidade tecnológica é um ecossistema vibrante de colaboração e aprendizado mútuo. Grande parte desses projetos e ideias — seja por meio de documentação, conteúdos educativos, projetos de grande complexidade ou listas para praticar programação — está centralizada na plataforma GitHub. Nessa plataforma, é possível colaborar com qualquer projeto, seguindo regras específicas de contribuição.

Em 2023, o GitHub lançou o **Innovation Graph**, uma ferramenta que permite visualizar, em um contexto mais amplo, as contribuições realizadas em repositórios da plataforma. Ela mostra dados abertos e *insights* sobre o impacto global e local dos desenvolvedores.

O *Innovation Graph* inclui métricas longitudinais sobre desenvolvimento de software para economias ao redor do mundo. Lançado hoje com uma página web e repositório dedicados, ele fornece dados trimestrais sobre *pushes* do Git, desenvolvedores, organizações, repositórios, idiomas, licenças, tópicos e colaboradores da economia. Desenvolvedores podem ter acesso à plataforma para identificar tendências ou encontrar projetos com os quais contribuir. A plataforma está disponível para consultas e para download dos dados através do link https://innovationgraph.github.com/.

O **Stack Overflow** é uma comunidade global de perguntas e respostas para programadores. Lançado em 2008, o site ajudou milhões de pessoas a encontrarem respostas de dúvidas

relacionadas à programação. Foi criado pelos desenvolvedores americanos Jeff Atwood e Joel Spolsky.

Em um artigo publicado no blog do Stack Overflow intitulado *"Five years ago Stack Overflow launched. Then a miracle occurred"* (Cinco anos atrás, o Stack Overflow foi lançado. Então, um milagre aconteceu.), os fundadores descrevem como surgiu a ideia de criar uma plataforma global que pudesse ajudar programadores do mundo inteiro. O texto do artigo afirma: "É por programadores, para programadores, com a intenção final de aumentar coletivamente a soma total de bons conhecimentos de programação no mundo. Não importa qual linguagem de programação você usa ou qual sistema operacional você chama de lar. Melhorar a programação é o nosso objetivo." (Stack Overflow Tech Blog, 2013, traduzido de https://stackoverflow.blog/2013/09/16/five-years-ago-stack-overflow-launched-then-a-miracle-occurred/).

Na plataforma, os usuários interagem diretamente, seja fazendo perguntas ou respondendo dúvidas. O Stack Overflow concede pontos aos usuários por responder perguntas e, em menor medida, por formulá-las. Essa dinâmica de interação é o que mantém a "engrenagem" do Stack Overflow funcionando.

Mas por que as pessoas dedicam tempo de suas vidas ocupadas para responder perguntas de outras pessoas, muitas vezes sem receber nada em troca além de pontos da plataforma? A resposta é simples: as pessoas da comunidade técnica gostam de cooperar e ajudar. Além disso, como já vimos, essa forma de disseminar conhecimento também traz aprendizado para quem ajuda, consolidando o próprio entendimento sobre os temas abordados, sem contar que alguns compartilham esse perfil em suas redes profissionais.

O **Dev.to** (https://dev.to/) é outra plataforma online gratuita e de código aberto para desenvolvedores. O Dev.to oferece uma experiência semelhante à de mídia social, onde os desenvolvedores

podem criar contas, comentar, reagir a artigos e também publicar conteúdos.

Normalmente, os desenvolvedores escrevem artigos técnicos sobre linguagens de programação, ferramentas, tutoriais etc. Já a plataforma **Daily.dev** (https://daily.dev/) é uma rede profissional que oferece notícias, tutoriais e fóruns para que desenvolvedores possam aprender, colaborar e crescer.

O Daily.dev é composto por uma extensão de navegador que oferece notícias personalizadas, com artigos e tutoriais de mais de 1.300 fontes, grupos e fóruns para conversar com outros desenvolvedores e se manter atualizado das últimas ferramentas e tecnologias do mercado.

Eventos presenciais

Eventos presenciais, por vezes, são mantidos e criados pelas comunidades técnicas. Essas comunidades geralmente são formadas e mantidas por desenvolvedores de software, engenheiros(as), entusiastas de hardware, designers, empresários, pesquisadores ou pessoas que tenham afinidade com a tecnologia. Elas dedicam seu tempo e disposição para manter a comunidade ativa, com o objetivo de compartilhar conhecimento, incentivar pessoas a palestrar, ensinar ou divulgar projetos.

Muitas das grandes comunidades que hoje realizam eventos em diversas partes do país começaram pequenas e cresceram graças à colaboração ativa de seus membros. Comunidades como NodeJS, Python Brasil, PrograMaria, Desprograme, .NET SP e DevPR não apenas compartilham conhecimento e contribuem para o crescimento profissional, como também oferecem oportunidades de emprego e fomentam projetos que surgem nesses espaços.

No repositório do GitHub criado pela comunidade técnica, **Agenda Tech Brasil** (https://github.com/agenda-tech-brasil/agenda-tech-brasil), é possível encontrar eventos que estão ocorrendo durante o

ano. Além do repositório do GitHub, é possível obter as informações acessando o site (https://agenda-tech-brasil-site.js.org/). Vale lembrar que na maioria deles o acesso é gratuito.

Eventos online

Alguns eventos são transmitidos de forma online e podem ser encontrados em plataformas como LinkedIn, YouTube e Discord. Rodrigo Cardoso, mais conhecido como "Pokemon" nas redes sociais, é desenvolvedor de software e também organizador de comunidades técnicas. Ele desenvolveu o site **Eventos de TI Dev** (https://eventosti.dev/), que serve como um agregador de eventos de tecnologia pelo Brasil. Nele, você encontra eventos online ou presenciais de todos os tipos de tecnologia. É possível cadastrar um evento no site.

10.1 Como você pode contribuir com a comunidade técnica e elevar seu nível de conhecimento

Organizar eventos, mesmo que gratuitos, envolve custos — seja em tempo ou dinheiro. É necessário encontrar um local que acomode um grande número de participantes e providenciar alimentos para o momento de confraternização e networking. Contudo, o aprendizado é gratificante. Quando você se torna um membro ativo da comunidade e começa a engajar-se, adquire novas responsabilidades, ganha visibilidade, melhora sua comunicação e conhece pessoas que podem agregar valor à sua carreira. Existem diferentes formas para participar da comunidade técnica, e algumas delas você pode conferir a seguir:

Escreva artigos técnicos ou publique em fóruns

Compartilhe seu conhecimento escrevendo tutoriais, guias ou reflexões sobre temas que domina, ou até mesmo algum problema que já resolveu. Plataformas como Dev.to, Medium ou LinkedIn são ótimas para isso. Além de ajudar outras pessoas, você consolida seu próprio aprendizado.

Contribua com projetos open source

Encontre projetos que estejam alinhados com seus interesses e habilidades. Normalmente, esses projetos são divulgados em redes sociais como BlueSky (bolha dev) ou Tab News, um site focado na comunidade da área de tecnologia, destinado a debates e troca de conhecimentos por meios de publicações e comentários, criados pelos próprios usuários. No Tab News, sempre há postagens sobre projetos colaborativos procurando pessoas que possam atuar em conjunto.

Compartilhe seus projetos

Disponibilize seus projetos pessoais no GitHub ou em outras plataformas. O Discord, por exemplo, é uma ótima ferramenta tanto para estudo colaborativo quanto para divulgação de projetos e ideias, e ajuda pessoas que estão iniciando na área.

Traduza documentações ou conteúdos

Muitos projetos *open source* precisam de ajuda para traduzir documentações para outros idiomas. Essa é uma forma valiosa de contribuir, especialmente para quem está começando. Além de cooperar com a comunidade, traduzir documentações é uma ótima forma de praticar inglês, um hábito essencial para profissionais de tecnologia, como discutido no capítulo anterior.

Mentorar outros desenvolvedores

Uma ótima forma de colaborar com a comunidade é oferecendo mentorias para iniciantes ou colegas que estão aprendendo algo

novo. Compartilhar sua experiência pode fazer uma grande diferença na jornada de outra pessoa.

Organizar ou ajudar em eventos

Além de engajar-se com a comunidade, você pode desenvolver suas habilidades de comunicação. Ofereça ajuda para organizar eventos ou meetups — isso pode incluir desde a logística até a curadoria de palestras e workshops.

Conversei com alguns profissionais da área para saber como a comunidade técnica ajudou em suas carreiras profissionais — e o que eles fazem para retribuir com o conhecimento adquirido.

"Acredito que escrever e compartilhar conteúdo na comunidade, e em muitos canais, contribuiu muito para meu desenvolvimento profissional, seja no ambiente corporativo, como nas comunidades, eventos. O objetivo de começar a escrever sempre foi ajudar o maior número possível de pessoas, fazer com que as histórias escritas no Medium/Substack servissem de alicerce para os novos profissionais do mercado de TI, atingindo as áreas de Linux, DevOps, automação, infraestrutura como código e Cloud.

Eu sempre soube que poderia ajudar as pessoas de alguma forma, e eu tive mais certeza ainda quando entrei na comunidade de TI. Foi quando eu conheci pessoas maravilhosas que também me ajudaram a vencer na profissão. Esse pessoal me ajudou demais — compartilharam conteúdo, dividiram conhecimento, e isso teve um impacto enorme em mim. Hoje, eu tenho o imenso prazer em ajudar aqueles que estão começando na área Tech.

E se vale uma dica, participem de eventos de TI, sejam presenciais ou online, isso agrega muito em termos de carreira, contato com a comunidade e aprendizado. Você vai aprender muito com profissionais excelentes da área, vai encontrar aquele amigo da empresa antiga, vai fazer muitos contatos e ver que os eventos da comunidade são sensacionais.

Depois que a gente passa a ter mais contato com a comunidade e cria laços de amizade e de profissão, a forma que encontrei de retribuir foi ensinando e compartilhando tudo o que um dia a própria comunidade me proporcionou. Eu me tornei recentemente Hashicorp Ambassador e AWS Community Builder. Esses reconhecimentos são a chave da minha vontade de retribuir com a comunidade.

Eu venho desde meados de 2016 estudando, compartilhando e criando conteúdos sobre Linux, DevOps, Cloud, automação, infraestrutura e sou grato por sempre estar com a comunidade, rodeado de ótimas pessoas, bons amigos e colegas de trabalho. Aprendi muita coisa com cada um, e esse reconhecimento mostra que nunca é tarde para ajudar as pessoas — que sempre existe uma solução e que podemos evoluir como profissionais."

— Amaury Borges Souza, DevSecOps na CI&T e Professor MBA FIAP

"Participe e, se possível, contribua. Ao se engajar, você sai do papel de espectador. Isso vai lhe trazer muito mais experiência. Quando você vai submeter uma palestra ou ministrar um curso, por mais simples que o conteúdo aparente ser, você acaba estudando mais e até reaprendendo o assunto.

Muitas vezes me perguntaram coisas que me deram uma nova visão. Aprende-se muito mais ensinando. Eu demorei muito para participar de comunidades e considero que perdi tempo enquanto estava fora. Isso garante uma contribuição absurda para a carreira: networking, eventos, ferramentas, boas discussões... tudo vem mais rápido dentro das comunidades."

— Alex Almeida, Tech Lead na Bouvet

"Participar da comunidade Tech vai muito além de uma maneira de se manter atualizado. É uma maneira extremamente efetiva de fazer networking e, por incrível que pareça, virou pré-requisito para muitas vagas com cargos de maior senioridade. Afinal, é esperado que, quanto mais você avance em sua carreira, mais você se preocupe em repassar conhecimento. E praticar essa soft skill é

extremamente importante para seu progresso profissional. O mais relevante é ter em mente que cooperar com a comunidade não é só sobre palestrar; é escrever artigos, participar de meetups, ser ativo em canais do Discord etc. Existem diversas maneiras de demonstrar interesse e engajamento — basta escolher aquela com a qual você mais se identifica."

— Aline Souza, Engineer Manager no ifood

O conselho que sugiro a você que está iniciando na área ou já tem tempo suficiente de carreira e quer aprender coisas novas é melhorar seu portfólio: esteja conectado com a comunidade. Esse espaço em constante evolução, onde a colaboração e a diversidade desempenham papéis dinâmicos, lhe trará grandes benefícios.

Ao participar e colaborar ativamente, você não apenas ganha conhecimento técnico, como também constrói redes de apoio, troca de experiência e ajuda a moldar o futuro da tecnologia. Como vimos pelos depoimentos, expandir *networking* e melhorar *hard skills* e *soft skills* são os objetivos de muitos profissionais da área.

Capítulo 11
Hábito 6: Primeiro entenda os conceitos, depois consolide com a prática

Figura 11.1: Hábito 6: primeiro entenda os conceitos, depois consolide com a prática.

Existem duas formas de aprender programação: pela abordagem tradicional (conceito antes da prática) e a "mão na massa" (aprendizado pela experimentação e imersão).

Cursos de graduação em Tecnologia da Informação (TI) geralmente seguem a primeira abordagem — ensinam a teoria, como lógica de programação, estrutura de dados, modelagem de banco de dados, como um compilador traduz o código-fonte para a linguagem de máquina, arquitetura de computadores e assim por diante. Isso garante aos estudantes uma base sólida de conhecimento teórico, aplicável a situações práticas e complexas.

Já os *bootcamps* ou métodos autodidatas invertem essa ordem. Começam por projetos práticos e vão introduzindo a percepção teórica conforme a necessidade. Ambas têm valor, porém, ao entender profundamente a base, os desenvolvedores evitam tornar-se reféns de soluções prontas, sem compreensão real.

Por exemplo, saber o funcionamento de alocação de memória para diferentes tipos de variáveis permite identificar vazamentos de memória e otimizar uso de recursos, garantindo que programas funcionem bem, além de evitar problemas de esgotamento. Quando desenvolvedores compreendem seu funcionamento, conseguem otimizar a eficiência do sistema e encontrar gargalos especialmente em sistemas críticos que exigem alta performance. Isso se aplica ao entendimento sobre normalização de dados: é imprescindível que o(a) desenvolvedor(a) tenha total percepção sobre o assunto para manter a consistência dos dados e garantir que as informações sejam precisas e confiáveis, o que torna o banco de dados mais robusto e sustentável.

Em uma época em que somos bombardeados por informações o tempo inteiro e mudanças constantes, a área de TI não é diferente. Nesse campo, as mudanças são tão rápidas que exigem que profissionais tenham constante atenção a ferramentas emergentes. No entanto, muitos desenvolvedores iniciantes podem perder o foco nos fundamentos em meio à avalanche de novidades.

É preciso focar no que é realmente necessário aprender, filtrando os conteúdos e informações que estão sendo absorvidas. Isso envolve dominar os princípios básicos e questionar-se constantemente sobre quais métodos são mais eficientes.

Por exemplo, aprender lógica de programação é o primeiro passo, pois ela pode ser aplicada em qualquer linguagem. Outro exemplo é compreender como aplicações web se comunicam e conhecer os protocolos que definem o tráfego de dados entre cliente e servidor. Da mesma forma, entender a normalização de bancos de dados é fundamental antes de aprender a sintaxe do SQL, pois trata-se da base teórica mais importante nesse processo.

O domínio dos princípios básicos capacita desenvolvedores a atuar com maior autonomia, diagnosticar problemas com precisão, implementar soluções robustas e adaptar-se a diferentes contextos técnicos. Essa base sólida contribui com a evolução de habilidades

que vão além de ferramentas e linguagens específicas. Ela permite que profissionais decifrem sistemas legados, promovam a refatoração de código de maneira mais inteligente, identifiquem a raiz dos problemas, façam correções de falhas críticas, antecipem vulnerabilidades e desenvolvam resiliência técnica para cenários imprevistos.

Seja curioso(a). A curiosidade nos leva a questionamentos como: "Por que devo fazer desta maneira e não de outra?" ou "Por que o fluxo segue esse caminho e não aquele?". Combine conhecimento teórico com questionamento constante — é assim que alguém que apenas executa tarefas se transforma em um solucionador de problemas.

Compartilho, a seguir, algumas estratégias que podem ajudá-lo(a) a desenvolver suas habilidades na base conceitual que todo bom desenvolvedor precisa ter.

Adote a mentalidade "Porque sim não é resposta"

Não aceite o "sim" com facilidade se não houver mais respostas concretas. Pergunte-se: "Por que esta solução funciona?", "Quais problemas ela resolve?" ou "Quais são as alternativas?". Se uma coisa não faz sentido pra você, se questione mais e busque respostas. Antes de copiar um código do *Stack Overflow*, questione-se por que usá-lo e se ele faz sentido para o seu problema.

Utilize Benchmarking para quase tudo

Benchmarking em computação é o ato de executar um programa de computador ou um conjunto de operações com o objetivo de avaliar o desempenho e ter resultados concretos sobre teorias. Basicamente, a ideia é comparar o desempenho de dois ou mais programas que fazem a mesma coisa.

Por exemplo, já precisei criar *benchmarking* para provar uma ideia que queria aplicar no código, só que, antes de sugerir e compartilhar essa ideia com o time, eu precisei testá-la para validar a minha teoria.

No GitHub, há um repositório chamado **BenchmarkDotNet** que ajuda a transformar métodos em *benchmarks*, rastrear desempenho e compartilhar experimentos de medição reproduzíveis. Você pode conferir no link https://github.com/dotnet/BenchmarkDotNet. Este, em específico, é focado na plataforma .NET (dotnet). Entretanto, há diversas páginas e repositórios no GitHub de *benchmarks*, diversas linguagens e frameworks, tais como **The Rust Performance Book** (https://nnethercote.github.io/perf-book/benchmarking.html) e **Google Benchmark** (https://google.github.io/benchmark/).

O *Benchmarking* pode ajudar você a identificar gargalos, prever falhas, avaliar performance de concorrência e validar sua teoria. Além do mais, é uma ótima abordagem para entender a fundo como as coisas funcionam por baixo dos panos.

Formule hipóteses

Formular hipóteses só é eficaz quando se entende profundamente o funcionamento da tecnologia envolvida. Quando enfrentar um problema, comece a formular hipóteses específicas baseadas nesses fundamentos. Quanto mais alinhadas aos conceitos, mais precisas serão suas investigações.

Por exemplo, se uma API (*Application Programming Interface*) retorna erro 500 em uma determinada requisição, seu conhecimento por aplicações *web* e arquiteturas distribuídas pode levar à hipótese: "isso pode ser um *timeout* ou resposta inválida de um outro serviço dependente". Da mesma forma, ao se deparar com uma consulta SQL lenta em tabelas grandes, entender como índices funcionam sugere imediatamente que "falta um índice nas colunas filtradas pelo *where*, forçando o banco de dados a varrer toda a tabela".

Transforme a teoria em analogias

Conceitos abstratos ficam mais claros quando comparados ao mundo real. Por exemplo, pense em um banco de dados como uma biblioteca: tabelas são como livros que armazenam informações, índices funcionam como catálogos que aceleram buscas, e consultas SQL equivalem a pedir ao bibliotecário para encontrar informações específicas. Da mesma forma, uma API REST opera como um restaurante, seus endpoints são o cardápio que lista opções disponíveis, requisições são como pedidos dos clientes, e respostas correspondem aos pratos servidos.

A ideia está em criar comparações que preservem a essência do conhecimento técnico. Isso evita o entendimento simples como "*cloud computing* é como uma nuvem". Em vez disso, utilize analogias funcionais para entender como *cloud computing* funciona. Por exemplo: podemos comparar *cloud computing* a um serviço de fornecimento de energia: você contrata e paga apenas o que consome. Por trás deste serviço há uma grande infraestrutura para atender os consumidores.

Quando você começa a fazer analogias, as coisas ficam mais claras porque você repensa como elas funcionam e se aplicam a outros contextos.

11.1 Do entendimento à execução

Consolidar os conceitos aprendidos com a habilidade prática torna o aprendizado real. Podemos observar isso em diversas situações: quando estudamos estrutura de dados e precisamos implementar pilhas e filas; quando analisamos as diferenças entre listas lineares e

listas ligadas para escolher a estrutura ideal e resolver um determinado problema; ou até mesmo quando aplicamos métricas para avaliar a eficiência de um algoritmo em projetos reais. Em todos esses casos, é a aplicação prática que solidifica o conhecimento teórico, porque você verá o resultado do que está sendo aplicado.

Crie exercícios de fixação e resolva problemas em plataformas como **HackerRank** ou **FreeCodeCamp** para visualizar os conceitos em ação. Experimente aplicar diferentes algoritmos, entenda quando utilizar cada um, implemente a medição de desempenho e observe os resultados.

Outro exemplo prático é com banco de dados. Depois de entender como funciona o Modelo de Entidade-Relacionamento (MER), crie um banco de dados com várias tabelas interligadas.

Simule um e-commerce: crie tabelas de pedidos, produtos, usuários, itens de pedido, entre outras. Insira dados, execute consultas e observe o funcionamento das chaves primárias e estrangeiras, dos relacionamentos e dos índices.

Dicas para consolidar a prática

Mensure o seu grau de evolução criando métricas de progresso. Assim, você consegue enxergar o que conseguiu evoluir e preencher lacunas.

Vejamos algumas sugestões:

- Crie um diário de aprendizado: registre seus *insights* e dificuldades. Particularmente, gosto de usar a ferramenta **Notion** para me ajudar nesta organização;
- Aplicação básica: resolva problemas simples sem consultar material;
- Aplicação avançada: identifique e resolva problemas complexos;
- Inovação: crie soluções originais baseadas nos conceitos aprendidos;

- Estabeleça ciclos breves de feedback: compartilhe seu projeto ou código com outros desenvolvedores e aceite sugestões ou críticas construtivas;
- Revise conceitos regularmente: retorne aos fundamentos quando necessário — se entendeu bem, avance; caso ainda haja dúvidas revise todo o conteúdo.

11.2 Experiência de outros desenvolvedores

Para reforçar a importância desse equilíbrio entre teoria e prática, compartilho a seguir relatos de desenvolvedores experientes que demonstram como aplicaram esse princípio em suas carreiras:

"Sempre priorizei estudar os conceitos fundamentais da programação utilizando 'coding kata' (plataforma de prática de exercícios de programação). Realizar vários code kata ajudou a consolidar meu conhecimento através da prática e também melhorou minha capacidade de resolver problemas complexos."

— Manoel Vitor Brito Moura, Especialista em Engenharia de Software na TOTVS

"Conceitos e teoria são a base de uma boa programação, mas é somente na prática que podemos realmente solidificar e absorver conceitos e transformá-los em habilidades sólidas e aplicáveis. Essa combinação nos torna desenvolvedores completos."

— Bruno Fecchio Salgado, Sênior Software Engineer e Tech Lead na Infojobs

"Certo dia, me aventurei na quadra de tênis, movido pela curiosidade. Sem nunca ter pegado em uma raquete, enfrentei jogadores experientes. A bola amarela saltava, era desafiadora; e eu

buscava cruzá-la para o outro lado. No entanto, a falta de fundamentos e treinamento me deixou frustrado. A experiência não foi completa. Percebi que essa situação ecoa no mundo do desenvolvimento de software. Copiar códigos da internet não é suficiente para compreender os conceitos. Estudar incessantemente, sem aplicação prática, também não é o caminho. Declarar uma classe abstrata pode ser fácil, mas entender por que ela deve ser usada? Isso requer mais do que teoria: exige prática.

Conceito e prática são como duas faces de uma mesma moeda, devem caminhar juntos. Assim como na quadra de tênis, onde cada movimento tem propósito, no desenvolvimento de software cada linha de código é uma tacada precisa. Aprendizado completo requer essa harmonia."

— Alex Almeida, Tech Lead na Bouvet

Assim como Manoel, Bruno e Alex destacaram, a combinação entre teoria e prática é fundamental para se tornar um(a) desenvolvedor(a) completo(a). Eu gosto de seguir uma abordagem semelhante e praticar lógica de programação utilizando plataformas de treino como HackerRank, LeetCode, Codecademy ou FreeCodeCamp. Primeiro, leio o conceito no livro; depois, gosto de visualizar alguns exemplos e só então consolido na prática usando essas plataformas para fixar o conhecimento da estrutura de dados que estou aprendendo.

CAPÍTULO 12
Hábito 7: Alinhe suas habilidades técnicas ao entendimento do negócio

Figura 12.1: Hábito 7: alinhe suas habilidades técnicas ao entendimento do negócio.

O trabalho de um(a) desenvolvedor(a) de software não se resume a utilizar as últimas tecnologias do mercado ou a construir aplicações visualmente atraentes. Na verdade, o papel de um(a) desenvolvedor(a) de software é resolver problemas de negócio usando a tecnologia como um meio. Isso exige não apenas o domínio de ferramentas e linguagens de programação, mas também uma compreensão profunda dos objetivos e desafios para os quais a solução está sendo criada.

Desenvolvedores que alinham suas habilidades técnicas ao entendimento das necessidades do negócio tornam-se ativos valiosos para suas equipes e empresas. Essas pessoas não só entregam soluções que atendem a demandas técnicas, como também contribuem diretamente para a geração de valor do produto ou serviço.

Antes de explorar a relação entre o software e o negócio, é importante entender o que, de fato, é um software. Software é uma sequência de instruções escritas para serem interpretadas por um computador, e tem como objetivo executar determinadas tarefas.

Em sua essência, o software é projetado para resolver problemas, automatizar processos e facilitar atividades específicas de forma rápida, eficiente e precisa, seja para indivíduos, equipes ou organizações. O valor do software vai além do funcionamento técnico: ele deve melhorar processos, otimizar recursos e trazer inovação para a maneira como vivemos e trabalhamos.

Um software de qualidade não é aquele desenvolvido com a "linguagem da moda" ou pelas tecnologias mais avançadas, e sim pela sua capacidade de transformar ideias em soluções reais e atender às necessidades do negócio. Um bom software entrega valor, promove o uso inteligente dos recursos e está alinhado aos objetivos estratégicos da empresa. Por isso, é fundamental que você, como um(a) desenvolvedor(a), compreenda o impacto do seu trabalho no contexto empresarial.

Como mencionei anteriormente, nas metodologias ágeis, cerimônias como planejamento do *sprint*, *dailies* e retrospectivas são momentos críticos para a equipe alinhar prioridades, identificar impedimentos, estimar tarefas, revisar processos e sincronizar ações com objetivos do negócio. Enquanto as cerimônias ágeis garantem o alinhamento operacional, o verdadeiro diferencial para um(a) desenvolvedor(a) está em transcender as tarefas técnicas e compreender profundamente o negócio que sustenta o produto.

Uma das grandes vantagens de ser um(a) desenvolvedor(a) é a possibilidade de trabalhar com diferentes produtos e conhecer diferentes tipos de negócios, entendendo como funcionam os processos e serviços.

Ao longo de minha carreira, já atuei em diferentes segmentos, como o setor financeiro, *e-commerce*, empresas que analisam interações

com clientes, entretenimento e também nichos mais específicos, como produtos voltados para demandas particulares. Veremos a seguir algumas dicas que você pode considerar para aprofundar melhor seu entendimento do negócio.

Só o onboarding não basta

Quando um(a) colaborador(a) inicia seu processo em uma nova empresa, é comum fazer o processo de *onboarding*, que nada mais é do que o processo de integração. Além de informações sobre o tipo de contratação, horários, salário e apresentação da equipe, neste processo há a apresentação do negócio da empresa em que ele(a) vai atuar.

Essa é a parte mais importante do *onboarding*: é o momento em que o(a) desenvolvedor(a) conhecerá o negócio em que vai trabalhar e, é claro, surgirão muitas dúvidas.

É importante que você busque o conhecimento do negócio, leia documentações, veja vídeos, procure conversar muito bem sobre o negócio com as pessoas do seu time. Aproveite o máximo do tempo de *onboarding*, que dura normalmente uma ou duas semanas dependendo do nível de complexidade do negócio, busque entender como o produto e serviço funcionam e quais os ganhos para a empresa.

Mapear o fluxo das aplicações

Durante esse tempo de *onboarding*, será apresentada também a parte técnica. Quais ferramentas utilizar e em quais aplicações você vai mexer. Aproveite esse momento e peça a alguém do seu time o desenho de arquitetura das aplicações. Peça a um membro do time que explique como os serviços e as aplicações se comunicam, quais são suas dependências e sua importância para o negócio.

Navegue pela aplicação, faça anotações, faça reuniões com algum membro que explique todo o processo. Grave essas reuniões. Tire algum tempo para revisar, repassar o fluxo do conhecimento e questioná-lo. Esteja sempre curioso(a) e demonstre interesse em entender o negócio.

Crie hábitos de perguntar e compreender

Se ficou com alguma dúvida, anote tudo e repassar esse entendimento com um membro do time. Se há ainda algo que não ficou claro, converse novamente e sanar essas dúvidas. É importante que as regras estejam claras. Isso se tornará mais fácil ao longo do desenvolvimento, então não fique preocupado(a) em querer absorver todo o negócio durante o tempo de *onboarding*. Isso pode gerar mais ansiedade e frustração por não entender todo o negócio de ponta a ponta. O importante é estar sempre envolvido(a), afinal entender todo o negócio exige dedicação e tempo.

Utilize code review para analisar código e entender as regras de negócio

Aproveite o processo de revisão de código e entenda as regras de negócio. Leia atentamente o enunciado da tarefa que foi passada ao desenvolvedor anterior, entenda o que foi pedido e o que foi feito.

Tente entender o fluxo da regra e tente reproduzi-la em seus testes. Se houver dúvidas, converse com o(a) desenvolvedor(a) responsável pela tarefa e tente sanar as dúvidas, procurando sempre estar atento(a) às regras. Explicarei mais sobre a revisão de código no capítulo seguinte.

Estabeleça ponte com Product Managers ou POs

Converse muito com os *Product Managers (PMs)* ou *Product Owners (POs)*, pessoas responsáveis pelo negócio. Participe de sessões sobre o negócio e anote os pontos importantes da conversa.

Entenda qual a visão da empresa, o que agrega valor nesse momento, quais são as prioridades e a atenção necessária ao negócio.

O diferencial de um(a) verdadeiro(a) desenvolvedor(a) está em compreender o negócio por trás do código

Quando você alia conhecimento técnico à visão estratégica, deixa de ser apenas um(a) executor(a) de tarefas e passa a ser um(a) parceiro(a) indispensável capaz de antecipar problemas, propor soluções inovadoras e entregar valor real.

Vale a pena lembrar: tecnologia é o meio, mas o fim sempre será resolver problemas humanos e impulsionar resultados. Invista nesse equilíbrio e seu impacto profissional será imensurável.

Entrevistei alguns desenvolvedores para entender a opinião deles sobre esse assunto.

"Habilidades técnicas são essenciais e ajudam a otimizar o tempo de desenvolvimento, mas entender o negócio também é necessário. A sinergia entre competências técnicas e entendimento do negócio agrega valor ao nosso trabalho e nos torna um parceiro estratégico dentro da equipe e/ou empresa."

— Bruno Fecchio Salgado, Sênior Software Engineer e Tech Lead na Infojobs

"Sempre me dediquei à parte técnica: frameworks, linguagens e arquiteturas. Acreditava que dominar essa "sopa de letrinhas" me tornaria um profissional exemplar. Parcialmente certo estava eu. No entanto, percebi que o conhecimento técnico, por si só, não basta. É como estar na metade do caminho.

A compreensão do negócio é o complemento essencial. Entender o propósito do software, seu público-alvo e as nuances do contexto resulta em intervenções e decisões mais precisas. Priorizar a performance, escolher quais funcionalidades entregar primeiro e até

mesmo gerenciar o débito técnico requerem essa visão holística. Encontrar material sobre o negócio nem sempre é fácil. É aí que entram os especialistas do domínio. Combinando técnica e conhecimento do negócio, você se tornará um(a) profissional altamente qualificado(a).”

— Alex Almeida, Tech Lead na Bouvet

“Um movimento muito recente que estamos observando no mercado é de desenvolvedores(as) que vão além do código, deixando para trás a fama de batedores de tecla. São profissionais preocupados com métricas, impacto no negócio, em participar de todo o processo de desenvolvimento, desde a priorização do backlog. Saber codificar, hoje, é o mínimo que se espera de um(a) desenvolvedor(a).”

— Aline Souza, Engineering Manager no ifood

“A coisa mais importante que as pessoas desenvolvedoras precisam absorver é que entender o negócio é crucial. O meio de resolução do problema importa muito, mas só é possível resolver problemas que entendemos pois, nesses casos, tentativa e erro podem custar caro para as empresas. Entregar código fácil de manter e sistemas que sejam fáceis de compreender é a chave do sucesso para a carreira.”

— Maurício Antunes, Senior Software Engineer na Clerk.com

Como esses profissionais demonstraram, quando você mescla o código ao contexto, deixa de ser um mero recurso técnico e se torna um acelerador de resultados. O mercado já valoriza profissionais que possuem essa visão estratégica. Mais do que codificar, é sobre saber por que e para quem você está desenvolvendo.

CAPÍTULO 13
Hábito 8: Lições que você pode aprender e ensinar durante o code review

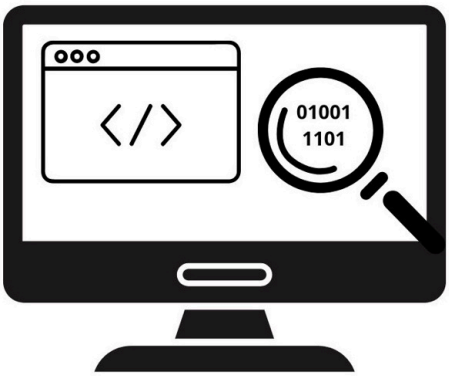

Figura 13.1: Hábito 8: Lições que você pode aprender e ensinar durante o code review.

Antes de se aprofundar nas lições valiosas que podemos aprender e ensinar durante o *code review*, é importante entender o que é essa prática e por que ela é tão fundamental no desenvolvimento de software.

O *code review*, ou revisão de código, é o processo no qual um(a) desenvolvedor(a) revisa o código escrito por outra pessoa do time, avaliando se segue os padrões técnicos, boas práticas e requisitos do negócio. O objetivo não é apenas identificar possíveis falhas ou inconsistências, mas também elevar a qualidade geral do software.

Uma *pull request* é uma proposta para mesclar as alterações de um *branch* em outro. Em uma *pull request*, os colaboradores podem revisar e discutir o conjunto de alterações proposto antes de integrá-las à base de código principal. As *pull requests* exibem as diferenças ou comparações entre o conteúdo no *branch* de origem e aquele no *branch* de destino.

Na prática, após o desenvolvimento da tarefa, o(a) desenvolvedor(a) abre um *pull request* e o(a) desenvolvedor(a) que estiver disponível poderá assiná-lo e fazer a revisão. Durante o refinamento técnico, o time estima o prazo da tarefa, incluindo o tempo necessário para revisão de código, que pode variar conforme o número de revisores necessários. Por exemplo, há projetos que exigem aprovação de dois ou mais membros do time.

Mais do que um processo de validação, o *code review* é uma oportunidade de aprendizado mútuo. No decorrer das revisões, são frequentes debates sobre os padrões de design, boas práticas, performance, legibilidade e até mesmo regras de negócio, a fim de validar se o(a) desenvolvedor(a) está implementando exatamente o que está sendo descrito na tarefa. Detalhes importantes podem passar despercebidos no momento da implementação, e é de responsabilidade do(a) revisor(a) evitar que isso passe adiante.

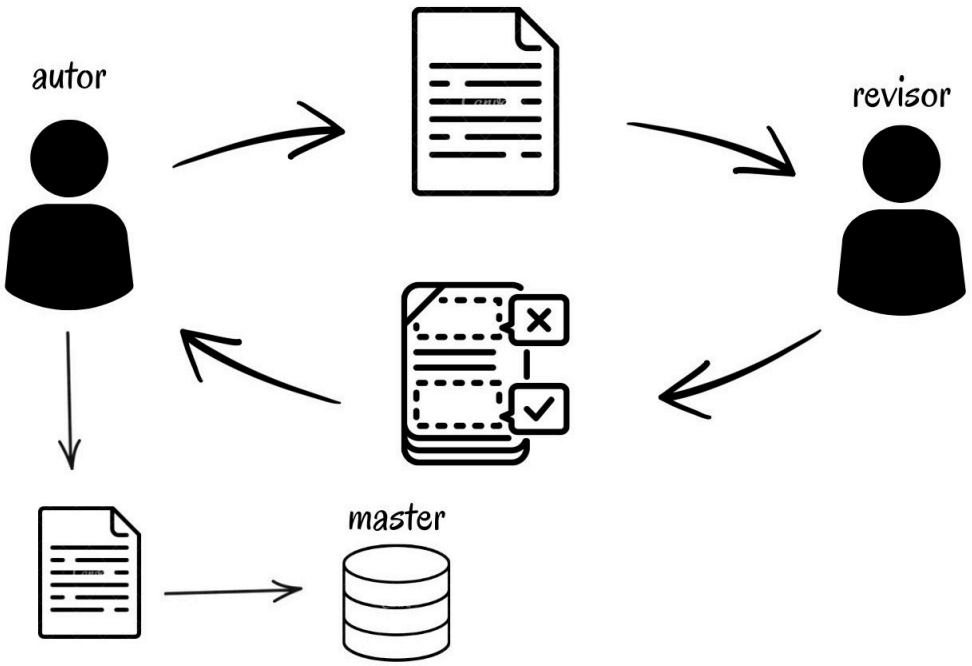

Figura 13.2: Exemplo de code review.

No entanto, nem sempre a revisão é algo puramente técnico. É comum que surjam divergências, muitas baseadas em preferências

pessoais, como estilo de escrita ou uso de determinadas estruturas da linguagem. Cada desenvolvedor(a) possui um estilo próprio, moldado por sua experiência e tendências individuais. Alguns são mais objetivos e pragmáticos; outros, mais criteriosos e atentos a cada detalhe da sintaxe.

Embora diferentes perspectivas enriqueçam a base de código e tragam visões valiosas sobre a funcionalidade como um todo, isso pode também tornar o processo de revisão um pouco mais lento. Já vivenciei revisões extremamente rápidas, que mais me pareceram uma aprovação automática, o que me fez duvidar da real análise do meu código. Por outro lado, já passei por revisões minuciosas, em que cada linha de código foi examinada, resultando em sugestões preciosas, melhorias significativas e prevenção de uma regra aplicada de forma incorreta.

Gostaria de ressaltar um ponto importante: é preciso equilíbrio e bom senso. A revisão de código pode ser minuciosa, mas também precisa ser objetiva. O foco da revisão deve estar no escopo da tarefa, no que foi proposto e desenvolvido, e não em reescrever outras partes do código que não fazem parte do escopo, a menos que haja uma justificativa técnica e clara para isso.

Uma revisão excessivamente detalhista que não impacta diretamente no resultado pode atrasar a entrega e desgastar a relação com o time. Já ouvi relatos de revisões de código terem o processo bloqueado durante dias devido a detalhes que estavam fora do escopo da entrega e nenhum dos lados chegava a um acordo. Quando há situações como essa, é bom envolver uma terceira pessoa, desenvolvedor(a) ou liderança técnica para tomar uma decisão que seja boa para os dois lados.

O debate sobre *velocidade versus profundidade* de revisão de código tem sido tema de artigos e estudos em empresas como a Google e Meta. O Google **Engineering Practices**, por exemplo, que serve como um guia de boas práticas para revisão de código, recomenda encontrar um ritmo bom na revisão, nem tão rápido que perca a

eficácia; nem tão lento que se criem gargalos. E ressalta: as revisões devem ser rápidas e otimizar a velocidade com que a equipe possa produzir o produto em conjunto (https://google.github.io/eng-practices/review/reviewer/speed.html).

Já o time de engenharia da Meta descreve as melhorias no tempo de *code review*, mostrando como o processo se tornou eficiente e como aumentou a produtividade sem sacrificar a qualidade de código. Além disso, descreve a utilização de um aprendizado de máquina: "Usamos aprendizado de máquina para identificar um diff que o revisor atual provavelmente desejará revisar. Em seguida, apresentamos esse diff ao revisor após ele concluir sua revisão de código atual. Facilitamos a navegação entre os possíveis próximos diffs e removemos rapidamente o revisor, caso um diff não seja relevante para ele. Após seu lançamento, descobrimos que esse recurso resultou em um aumento geral de 17% nas ações de revisão por dia (como aceitar uma diferença, comentar etc.) e que os engenheiros que usam esse fluxo realizam 44% mais ações de revisão do que o revisor médio!". (Trecho traduzido artigo oficial da Meta em https://engineering.fb.com/2022/11/16/culture/meta-code-review-time-improving/).

Veja algumas dicas que você pode pôr em prática para aprender e ensinar durante o *code review*.

Se é algo novo ou ficou na dúvida, pesquise

Se durante a revisão você identificou uma estrutura da linguagem que não conhecia, ou teve dúvidas sobre determinada funcionalidade do código, pesquise. Procure entender por que foi feito daquela forma, se é algo novo na estrutura da linguagem. Veja outros exemplos de como aplicar e procure entender para que você possa utilizar em seu código.

Utilize a revisão de código para aprender ou ensinar

Como descrevi anteriormente, a revisão de código é um aprendizado em conjunto. Durante a revisão, os pares (autor e revisor) trocam conhecimento. Você pode ensinar algo novo para uma pessoa durante a revisão ou aprender com quem está fazendo a revisão do seu código, seja de boas práticas ou mesmo de estruturas de código.

Por exemplo, há um conceito interessante que aprendi neste processo: o revisor do meu código me ensinava da importância de "quebrar o código" em partes menores, e durante essa conversa ele me recomendou livros sobre refatoração, me indicou exemplos e mostrou como deixar o código mais legível.

O que você deve procurar em uma revisão de código

Alguns pontos importantes que você deve observar durante o *code review* são:

- Design: isso se integra bem com o restante do sistema e as interações entre diferentes componentes?
- Funcionalidade: isso funciona como o esperado?
- Complexidade: este código está mais complexo do que deveria ser?
- Nomenclatura: a nomenclatura das variáveis, métodos, classes estão claras?

Teste a funcionalidade e valide as regras de negócio

Durante a revisão, faça o teste da funcionalidade como um todo. Por exemplo: se a tarefa era incluir um cálculo novo de um imposto em determinada parte do sistema, após validar o código, faça o teste e veja se está funcionando adequadamente. Se ficou com dúvidas sobre o fluxo da funcionalidade, converse com o autor do código.

Boas práticas ao fazer uma revisão de código

Você pode seguir essas boas práticas não somente durante a revisão de código, como também ao enviar seu código para revisão:

- Tente revisar seu código: antes de enviar para seus colegas, tente lê-lo e entendê-lo. Procure por partes que possam ser confusas;
- Escreva uma descrição do que foi alterado: isso deve explicar quais mudanças foram feitas em alto nível e por que elas foram necessárias;
- Tente entender o que foi alterado: se necessário, leia várias vezes, classe por classe, linha por linha;
- Comente com gentileza: foque nas mudanças, formule perguntas ou sugestões e deixe pelo menos um comentário positivo. Por exemplo: "você não acha que fica melhor fazer dessa forma?";
- Aprove o *pull request* quando estiver bom o suficiente. Não busque a perfeição, mas mantenha um padrão elevado. Não seja excessivamente exigente;
- Nem sempre é erro — às vezes, é só estilo: desenvolvedores(as) diferentes têm estilos e abordagens distintas de resolver um problema. O fato de alguém ter feito algo de forma diferente da sua não significa que está errado. Avalie com cuidado: será que é mesmo um problema técnico ou apenas uma preferência sua?

Lidando com feedbacks e críticas

Algumas sugestões para lidar com feedbacks incluem:

- Não leve para o lado pessoal: críticas ao código não são críticas a você como profissional. Muitas delas são enriquecedoras para nosso crescimento profissional;
- Tenha a mente aberta: esteja aberto a dialogar e a aprender. É importante opinar e demonstrar interesse no processo;

- Reconheça bons pontos: se o feedback trouxe melhorias, agradeça e reconheça.

Aprendendo com diferentes perspectivas

Conversando com alguns desenvolvedores sobre suas experiências, pude coletar alguns comentários interessantes sobre o que eles mais valorizam no processo de *code review* e como lidam com situações desafiadoras durante as revisões.

"O code review é uma ótima oportunidade para aprender e ensinar boas práticas de codificação, incluindo convenções de nomenclatura, manter o código limpo e legível, evitar duplicações, documentar adequadamente e realizar refatorações quando necessário. Essas práticas contribuem para um código mais robusto e sustentável."

— Marcos de Medeiros, Software Developer na Bravium

"Valorizo muito a objetividade no code review. Às vezes, vejo situações em que o pull request está praticamente perfeito. No entanto, o revisor, apenas para demonstrar que fez alguma crítica, cria problemas onde não existem. Por exemplo, sugerir trocar um 'for' simples para um 'foreach' sem necessidade.

Já enfrentei conflitos durante revisões em que o analista e o tech leader não conseguiam chegar a um acordo. Nesses casos, recorri à documentação para demonstrar que ambas as abordagens eram válidas e que a alteração que o tech leader estava exigindo não agregaria valor real ao produto."

— Alvaro Camillo Neto, Senior Staff Engineer na F1RST Tecnologia

Parte III: O mercado e o futuro da carreira Dev

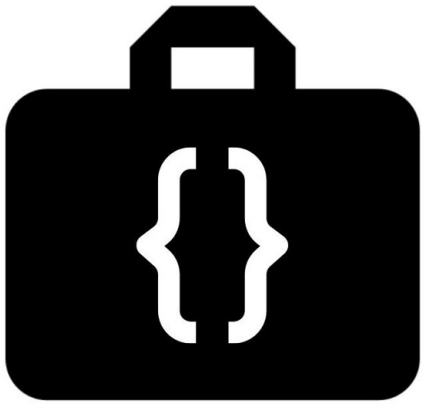

Figura 1: O mercado e o futuro da carreira Dev.

Capítulo 14
Tendências e investimentos em IA no mercado tech

Figura 14.1: Tendências e investimentos em IA no mercado tech.

A Inteligência Artificial (IA) tem sido um tema central nas empresas, basta olhar para os lados e perceber que ela está praticamente em tudo. A maioria das empresas que possuem um produto ou serviço está utilizando IA em seus processos para automatizar suas tarefas e ganhar produtividade.

A ideia é que pessoas possam trabalhar de forma mais estratégica e utilizar a IA para tarefas operacionais e generalistas. Por exemplo, empresas estão adotando ferramentas como **GitHub Copilot** para auxiliar a escrita de código que contribui com a produtividade dos desenvolvedores. Há outras ferramentas como **Chat GPT**, **Gemini** ou até mesmo **Notions AI** para auxiliar na revisão de textos, resumos de reuniões e organização de tarefas.

Uma pesquisa realizada pela **Spiceworks** em 2025 trouxe importantes percepções sobre gastos em TI, carreiras e tendências tecnológicas (Spiceworks, 2025, https://www.spiceworks.com/research/it-report/). O estudo revelou que as empresas estão planejando adotar novas tecnologias para

impulsionar seus resultados de forma positiva. Com a rápida disseminação de IA, os investimentos nesta área estão crescendo de forma significativa. O relatório destaca dados impressionantes: 64% das organizações planejam aumentar seus orçamentos de TI em 2025, enquanto 54% das empresas pretendem especificamente aumentar seus gastos com IA generativa.

É claro que o mercado está mudando rapidamente e tentando acompanhar as evoluções tecnológicas, sendo difícil prever com precisão o cenário dos próximos anos. Entretanto, é evidente que a IA veio pra ficar, e tanto empresas quanto profissionais precisarão se adaptar a essa nova realidade.

Além dos dados da **Spiceworks**, uma matéria publicada pela revista **Exame** em 2025 reforça essa tendência mostrando os grandes investimentos das *Big Techs* em IA para 2025 (Exame, Fevereiro 2025 https://exame.com/inteligencia-artificial/big-techs-devem-investir-us-320-bilhoes-em-ia-em-2025/?utm_source=copiaecola&utm_medium=compartilhamento). As gigantes da tecnologia como **Amazon**, **Microsoft**, **Google** e **Meta** devem investir 320 bilhões de dólares em IA em 2025.

De acordo com a matéria, a maior parte do investimento da **Amazon** será direcionada para o **Amazon Web Services** (serviços em nuvem da AWS) e à escalabilidade da infraestrutura de IA. Já o **Google** anuncia que planeja investimentos de cerca de 75 bilhões de dólares em 2025. Na **Meta**, os gastos com IA estão aumentando e a empresa projeta investimentos de cerca de 65 bilhões de dólares neste ano. A **Microsoft** anunciou que planeja investir 80 bilhões de dólares em infraestrutura de IA em 2025. A empresa entende que a demanda por serviços de IA está sendo cada vez mais requisitada. O **GitHub Copilot**, ferramenta desenvolvida pela empresa, conta também com o Copilot no Windows e no Microsoft 365.

Com investimentos dessa magnitude e o avanço acelerado das ferramentas de IA, cresce também a demanda por profissionais capacitados não apenas em programação, mas também em saber

como interagir de forma rápida e estratégica com essas ferramentas, como é o caso dos desenvolvedores que dominam a criação de *prompts* eficazes.

Um *prompt* nada mais é do que um texto em linguagem natural que solicita que a IA generativa execute uma tarefa específica. A IA generativa é uma solução de Inteligência Artificial capaz de criar conteúdos como textos, imagens, vídeos e músicas. Ela é alimentada por modelos grandes de *Machine Learning* (ML, aprendizado de máquina), que utilizam redes neurais pré-treinadas em um grande volume de dados. Os usuários podem interagir com esses modelos por meio de várias combinações de dados de entrada. São modelos poderosos e muito precisos, e quanto mais bem detalhado for o *input* — ou seja, a entrada desses dados sobre o que você quer e o que espera — mais assertiva é a resposta que você obtém.

Desenvolvedores com habilidades em prompts

A pesquisa da **Spiceworks** destaca o investimento em IA: as empresas estão buscando perfis de desenvolvedores que tenham conhecimento básico em *prompts*. Basta uma rápida busca por vagas no LinkedIn para perceber que algumas não apenas incluem a palavra *prompt* entre os requisitos, como também a trazem no próprio título da vaga.

Demanda por especialistas em Cibersegurança

Com o avanço das ferramentas de IA, a procura por especialistas em Cibersegurança aumentou. Hackers estão usando IA para modernizar seus ataques, e criminosos utilizam a tecnologia para criar *deepfakes* altamente realistas (vídeos, áudios e imagens manipuladas), realizar ataques de *phishing* personalizados, enviar e-mails falsos, entre outros. Com isso, a busca por profissionais especializados na área de segurança tende a crescer cada vez mais.

Ascensão dos Engenheiros de MLOps e Infraestrutura de IA

Empresas estão investindo muito na cultura do *Machine Learning Operations* (MLOps), conjunto de práticas para colaboração entre cientistas de dados e profissionais de operações. Em um projeto de *Machine Learning* (ML), cada papel tem sua responsabilidade.

O cientista de dados é focado em elaborar hipóteses e testá-las. Sua função também inclui analisar e interpretar grandes volumes de dados para identificar padrões que possam auxiliar nas tomadas de decisão. Já o engenheiro de dados é responsável por processar, armazenar e disponibilizar dados para cientistas e analistas de dados. Como podemos observar, são responsabilidades distintas e, mesmo com a pessoa engenheira de dados apoiando o cientista de dados em tarefas complexas de pipeline, ainda permanece uma lacuna aberta.

Esta lacuna surge justamente quando o projeto precisa ser automatizado e implementado no modelo ML. Da mesma forma que descrevi nos capítulos anteriores sobre o processo de desenvolvimento de *software*, para modelos ML isso não muda; o projeto precisa ser testado, implantado, monitorado e automatizado em um pipeline. É aí que entra o papel do MLOps, que aumenta a qualidade, simplifica processos de gerenciamento e automatiza a implantação de modelos ML e *Deep Learning* (DL).

Em um artigo da **Fortune Business Insights** é destacado o seguinte: "O tamanho do mercado global de MLOps foi avaliado em 1,58 bilhão de dólares em 2024. O mercado deve crescer de 2,33 bilhões de dólares em 2025 para 19,55 bilhões de dólares em 2032, exibindo um CAGR de 35,5% durante o período de previsão". (Abril, 2025, https://www.fortunebusinessinsights.com/pt/mlops-market-108986)

Na visão da CTO Ludmila Pontremolez, contratação vai além do técnico

Em um episódio do meu podcast ***Café Debug** (nº 163), conversei com Ludmila Pontremolez, CTO e cofundadora da Zippi, sobre as tendências do mercado de trabalho em 2025. Segundo ela, nos processos seletivos dos quais participa, a maioria dos desenvolvedores demonstra maior preocupação em dominar ferramentas técnicas do que em desenvolver habilidades de resolução de problemas ou compreender a lógica de negócios da empresa.

Durante nossa conversa, Ludmila destacou como as ferramentas de IA estão revolucionando o ambiente de trabalho nos times de desenvolvimento, acelerando a produtividade e automatizando tarefas rotineiras. Na área de gestão, elas também contribuem para a otimização de processos, como a organização e o resumo de atas de reuniões, a formatação de documentos e a realização de testes de novas soluções tecnológicas com o produto.

"Algo que a Zippi tem investido muito é nas ferramentas de IA. Nós testamos várias no nosso dia a dia, desde análise de dados e criação de queries até automatização de workflows e anotações de reuniões. E tem feito uma diferença enorme, especialmente nas entrevistas com clientes. Depois de cada call, eu pego os principais insights e tópicos discutidos, jogo na IA e faço perguntas bem direcionadas. Ela me ajuda a extrair resumos claros e identificar as palavras-chave mais importantes."

— Ludmila Pontremolez, CTO e cofundadora da Zippi

Além das fronteiras: Como a competição por IA está redefinindo o mercado global

O ano é 2025 e a "corrida das IAs" está apenas começando. Não basta que o modelo seja inteligente o suficiente para responder ao usuário. A velocidade tornou-se um sinal de força competitiva. O desenvolvimento das IAs tem gerado dois polos dominantes (EUA e

China), com abordagens distintas. E para criar modelos altamente eficazes e com menor tempo de resposta, é necessário investir pesado em *hardware*.

A evolução dessas tecnologias está profundamente ligada à produção de semicondutores e processadores de alto desempenho. Afinal, construir um modelo de IA avançado exige enorme capacidade de processamento. A empresa **NVIDIA** se destaca como líder global, sendo a principal fornecedora de GPUs e *hardware*.

No entanto, esse cenário vem se transformado com a chegada da *startup* chinesa **DeepSeek**, uma empresa de desenvolvimento de IA conhecida por seus modelos de linguagem de grande porte (LLMs) de código aberto. A DeepSeek "sacudiu" o Vale do Silício ao demonstrar que é possível ser mais eficiente mesmo operando com baixo custo de processamento, alcançando praticamente os mesmos padrões das principais empresas americanas.

Controvérsias à parte, esse modelo acendeu um alerta não apenas para a comunidade científica e tecnológica, mas também para o mercado. Ele quebra o paradigma dominante até então, no qual as *Big Techs* americanas e grandes corporações acreditavam que só era possível evoluir em IA com mais poder computacional e grandes volumes de dados para treinar esses modelos. A inovação chinesa provou que é possível pensar em outras estratégias eficazes.

Tendências de competências: investindo no profissional do futuro

À medida que essa evolução da IA avança com investimentos bilionários, o perfil do profissional tech se reconfigura completamente.

Os profissionais precisarão desenvolver as seguintes competências: proficiência tecnológica, visão estratégica de negócios e agilidade adaptativa. Como descrevi na parte 2 do livro, sobre o entendimento do negócio, resolver problemas de forma simples e utilizar a IA de

forma eficaz serão diferenciais essenciais para quem deseja se destacar neste novo cenário.

No próximo capítulo, vamos explorar melhor o que são os *prompts* e as ferramentas de IA generativa que podemos utilizar em nosso dia a dia, além de como extrair o melhor dos resultados por meio de entradas de texto.

CAPÍTULO 15
Ferramentas de IA que potencializam o trabalho dos devs

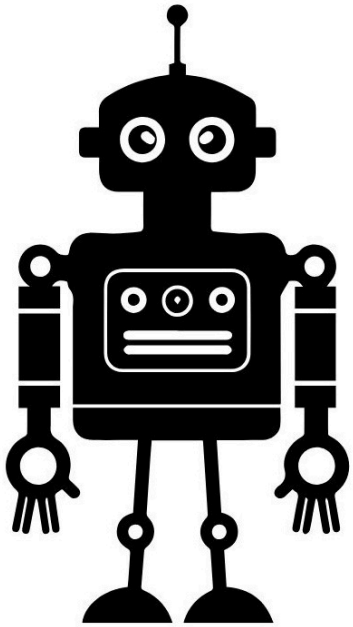

Figura 15.1: Ferramentas de IA que potencializam o trabalho dos devs.

A Inteligência Artificial potencializa nossa produtividade e, conforme apresentado no capítulo anterior, empresas estão investindo cada vez mais nessas plataformas e ferramentas para ampliar sua produção. Usada de forma consciente, a IA pode ajudar você a otimizar tarefas, verificar códigos, validar soluções, aprender conceitos ou ferramentas tecnológicas, auxiliar no *code review*, desenvolvimento de aplicações avançadas, análise de dados etc.

Isso exige que as pessoas desenvolvedoras tenham uma nova habilidade: a de escrita de *prompts*. Cada vez mais, isso é visto com bons olhos pelas empresas. Afinal, programadores que dominam a arte de criar *prompts* eficientes conseguem extrair o melhor das

respostas da IA, acelerando o trabalho e melhorando a qualidade do seu código.

Um *prompt* bem elaborado pode ajudar a gerar trechos de código, revisão de códigos, funções ou até estruturas completas em segundos. Além de poder automatizar tarefas repetitivas, *prompts* podem ser usados para criar scripts, comandos ou até documentações automaticamente.

Neste repositório do **GitHub** há uma lista de *prompts* poderosos para o **ChatGPT** que pode ajudar você a liberar sua criatividade: https://github.com/PickleBoxer/dev-chatgpt-prompts. Você encontrará dicas de engenharia de *prompts*, sugestões para criação de casos de testes e formas eficazes de detalhar melhor o contexto da sua pergunta para a IA.

Neste capítulo, vamos explorar ferramentas de IA generativa e algumas funcionalidades que elas entregam.

GitHub Copilot

O GitHub Copilot é uma ferramenta de Inteligência Artificial desenvolvida pelo GitHub em parceria com a OpenAI, com o objetivo de auxiliar usuários. Atende às IDEs (Ambientes de Desenvolvimento Integrado) **Visual Studio Code**, **Visual Studio**, **Neovim** e **JetBrains**, oferecendo sugestões de código para preenchimento automático. O Copilot possui a versão gratuita, com limitações, e a versão Premium com mais recursos.

- Copilot Chat: Uma interface de chat que permite fazer perguntas sobre a codificação. É possível fazer perguntas relacionadas ao seu código, pedir sugestões e melhorias. (https://docs.github.com/pt/copilot/using-github-copilot/copilot-chat/asking-github-copilot-questions-in-your-ide);

- Copilot CLI: Uma interface semelhante a um chat no terminal, em que você pode fazer perguntas sobre a linha de comando, além de solicitar que o Copilot forneça sugestões de comandos

ou explicações. (https://docs.github.com/pt/copilot/using-github-copilot/using-github-copilot-in-the-command-line);
- Revisão do GitHub Copilot: Sugestões de revisão geradas por IA para ajudar você a escrever um código melhor. Diferente do Copilot Chat, a solicitação é feita no Github durante a criação do *pull request*.(https://docs.github.com/pt/copilot/using-github-copilot/code-review/using-copilot-code-review#como-solicitar-uma-revis%C3%A3o-do-copilot);

- Copilot Pull Request Summaries: Resumos gerados por IA das alterações feitas em uma *pull request*, indicando quais arquivos foram afetados e em quais pontos o revisor deve se concentrar durante a revisão. Saiba mais em: https://docs.github.com/pt/copilot/using-github-copilot/using-github-copilot-for-pull-requests/creating-a-pull-request-summary-with-github-copilot

Amazon CodeWhisperer

O Amazon CodeWhisperer é um gerador de código de uso geral, com tecnologia de aprendizado de máquina, que fornece recomendações de código em tempo real. À medida que você escreve o código, o CodeWhisperer gera sugestões automáticas com base no código e nos comentários existentes.

Você pode utilizar o CodeWhisperer com as seguintes IDEs: **Visual Studio Code**, **Visual Studio**, **AWS Cloud9** e **JetBrains**. (https://docs.aws.amazon.com/codewhisperer/latest/userguide/what-is-cwspr.html).

Destaco duas funcionalidades do CodeWhisperer:

- Integração com AWS: O CodeWhisperer é especializado em desenvolvimento na nuvem da Amazon, oferecendo sugestões personalizadas tanto para codificação quanto para infraestrutura. Isso representa uma vantagem significativa para quem constrói aplicativos dentro do ecossistema da AWS;

- Verificação de segurança e sugestão de correção: CodeWhisperer está integrado ao **Amazon Code Guru** (https://docs.aws.amazon.com/codeguru/detector-library/), que é um serviço de ML da Cloud, projetado para automatizar revisões de código, identificar bugs e sugerir melhorias de desempenho.

Jetbrains AI

O JetBrains AI é um assistente de inteligência artificial integrado às IDEs e as ferramentas de codificação da JetBrains e do **ReSharper**.

Ele foi projetado para ajudar pessoas desenvolvedoras a escrever, entender e melhorar seu código de forma mais eficiente, podendo optar por um plano mensal ou anual disponível para clientes individuais e organizações (https://blog.jetbrains.com/pt-br/blog/2023/12/06/apresentando-o-jetbrains-ai-e-o-ai-assistant-nos-ides/#).

Funções que você encontra no Jetbrains AI:

- Chat com IA: Atua como um interlocutor sensível ao contexto dentro da IDE. O usuário pode interagir com o chat e perguntar, por exemplo, como implementar uma função, concluir uma tarefa em uma linguagem específica ou depurar um problema.
- Refatoração: Ajuda a melhorar a estrutura do código sem alterar o comportamento. Sugere otimizações e padronizações;
- Correção de bugs: Identifica possíveis erros e sugere correções;
- Redação de documentação: Gera comentários e documentação para funções e classes, no contexto de código.

Cursor

Cursor AI é um editor de código inteligente bem semelhante ao **VSCode**, que aproveita o poder da IA para auxiliar os desenvolvedores em todas as etapas do processo de codificação (https://www.cursor.com/features).

Principais recursos do Cursor AI:

- Terminal Ctrl K: Utilize Ctrl + K no terminal para digitar comandos em inglês simples. O cursor os converterá automaticamente no comando de terminal correspondente.
- Pergunta rápida: Se você tem alguma dúvida breve sobre determinadas partes do seu código, você pode selecionar e clicar em "pergunta rápida" para obter sua resposta imediatamente;
- Leitura de imagens: A ferramenta faz a leitura de imagens: você pode anexar uma imagem para a caixa de entrada e incluí-la no contexto visual do bate papo;
- Respostas da base de código: Utilize `@Codebase` ou `Ctrl Enter` para fazer perguntas sobre sua base de código. O cursor pesquisará sua base de código para encontrar código relevante para sua consulta;
- Loops em erros: O cursor pode detectar erros de lint automaticamente e aplicar correções, reduzindo a necessidade de depuração manual.

Chat GPT

O ChatGPT é um *chatbot* desenvolvido pela **OpenAI** que usa Inteligência Artificial para gerar respostas a perguntas em linguagem natural. Saiba mais em https://chatgpt.com.br/qual-e-a-funcao-do-chatgpt/.

O ChatGPT conta também com:

- Assistente virtual: Ele pode ajudar usuários a obter informações, resolver problemas e executar tarefas com base em comandos de texto;
- Geração de imagens: O recurso **DALL-E** cria imagens através de comandos de texto;
- Resumo de conteúdos do YouTube: O Video Summarizer é uma ferramenta que resume o conteúdo de vídeos do YouTube com base na transcrição automática;
- Geração de vídeos realistas: o Sora é uma tecnologia desenvolvida pela OpenAI que permite gerar vídeos realistas a partir de descrições em texto.

Ollama

Ollama é uma ferramenta de código aberto que permite executar modelos de *Large Language Models* (LLMs) diretamente no seu computador local. Isso a torna uma opção popular para desenvolvedores de IA, pesquisadores e empresas que priorizam o controle e a privacidade dos dados, pois evita a necessidade de depender de servidores externos para executar e utilizar os modelos.

No GitHub do Ollama, a documentação é clara e bem explicativa para quem deseja executar localmente (https://github.com/ollama/ollama).

Outras funcionalidades:

- Execução local dos modelos: Através da documentação do GitHub, você pode configurar o Ollama para rodar localmente na sua máquina, utilizando comandos no terminal;
- Customização com Modelfiles: Você pode criar seus próprios *Modelfiles*, que são arquivos de configuração do próprio Ollama utilizado para personalizar, criar e controlar o comportamento de

modelos de linguagem localmente (https://ollama.readthedocs.io/en/modelfile/);
- API Local: O Ollama expõe uma API local compatível com OpenAI, que permite gerar textos, conduzir conversas e receber respostas estruturadas em JSON.

Qualquer semelhança não é mera coincidência

O mercado atual oferece centenas de ferramentas de IA generativa, cada uma com suas peculiaridades.

Embora ferramentas como **Claude.AI**, **Deepseek** e **Gemini** compartilhem a base de chatbots generativos, cada uma apresenta um desempenho único em velocidade, precisão da informação e recursos. Utilize-as no trabalho para aumentar a produtividade nas suas tarefas.

CAPÍTULO 16
Como se preparar para entrevistas técnicas e comportamentais

Figura 16.1: Como se preparar para entrevistas técnicas e comportamentais.

O mercado de tecnologia passou por várias transformações significativas nos últimos anos. Os processos seletivos tornaram-se mais rigorosos e competitivos, exigindo que cada profissional demonstre não apenas expertise técnica, mas habilidades interpessoais e a capacidade de resolver problemas complexos sob pressão.

Se antigamente uma única etapa de entrevista técnica era suficiente, hoje é comum que profissionais enfrentem até seis etapas diferentes: desde *coding challenge* até apresentações sobre *system design*.

É importante ressaltar que cada organização possui seu próprio formato de contratação, com variações significativas na duração e na estrutura. Algumas avaliações podem se estender por semanas, envolvendo múltiplos processos eliminatórios, enquanto outros são concluídos em poucos dias, com avaliações concentradas.

Poucas pessoas sabem que não existe um "formato padrão" no mercado tech. Cada empresa desenvolve sua própria metodologia. Entender essas variações é crucial para sua preparação.

Já participei de muitos processos seletivos ao longo da minha carreira, e nem todos foram iguais. Criei "projetinhos" na primeira etapa, seguindo o enunciado que me foi enviado, respondi perguntas técnicas e apresentei soluções para problemas durante a fase técnica conversacional. Também precisei explicar o código que estava escrevendo durante a sessão de *live coding*.

Muitas pessoas se preparam focando apenas no *live coding* ou *system design*, mas na prática descobrem que existem outras cinco fases inesperadas. Para evitar essas surpresas, vou mapear os diferentes formatos que você pode encontrar e, após apresentar todos os tipos de avaliação que compõem as entrevistas, darei algumas dicas de como se preparar melhor para cada uma delas.

Live Coding

É um método comum que as empresas adotam para avaliar o nível de conhecimento técnico dos desenvolvedores. Durante essa fase, enquanto os candidatos codificam, são solicitados a depurar ou aprimorar o código em tempo real, normalmente por meio de um ambiente de codificação online ou IDE como **VSCode**.

Em alguns casos, certas empresas adotam essa prática utilizando o **Bloco de Notas** ou até mesmo um documento do **Word**. A ideia é apenas entender a lógica de pensamento do(a) desenvolvedor(a) e observar como a pessoa resolveria o problema. Para algumas organizações e avaliadores, esse formato oferece uma visão mais

prática e exige mais habilidades de programação do que um modelo tradicional de perguntas e respostas.

Para muitos profissionais, esse tipo de entrevista pode ser bem desconfortável. Como as pessoas ficam sob o olhar atento dos entrevistadores, isso gera um ambiente de pressão moderada. Mesmo que o(a) desenvolvedor(a) seja bem qualificado(a), pode cometer alguns erros que comprometem o resultado da sua aprovação.

A melhor forma de se preparar para o *Live Coding* é treinar sua comunicação e dominar estruturas de dados. Pratique a resolução de problemas em plataformas de escrita de código e, durante os treinos, tente explicar o que está fazendo e por quê — em voz alta, como se estivesse explicando para alguém. Você também pode gravar vídeos como se estivesse dando uma aula, apresentando a solução passo a passo.

Entrevistas de algoritmos

A técnica que utiliza algoritmos é bastante usada pelas FAANG, sigla que representa as *big techs* como **Meta** (antigo Facebook), **Amazon**, **Apple**, **Netflix** e **Google**. As companhias que aplicam esses testes procuram avaliar:

- A solução proposta: se ela funciona para todos os casos e motivos pelos quais foi escolhida;
- Os algoritmos e estruturas de dados essenciais: entrevistadores procuram avaliar qual domínio o(a) candidato(a) possui com estruturas de dados, tais como a complexidade de tempo e espaço Big(O), arrays, grafos, pesquisa binária, heaps e árvores;
- A comunicação: explicar o raciocínio durante a resolução;
- Debugging: identificar e corrigir erros.

A melhor estratégia para se sair bem é praticar, garantir uma resposta funcional, preocupar-se com casos extremos e refatorar o código em segundo plano. Essas questões são independentes da linguagem, então você pode utilizar uma linguagem de programação de alto nível e não tipada, como **Python**, ou optar por aquela com a qual estiver mais familiarizado(a) e confortável.

Há diversos canais no **YouTube** ou na **Twitch** que abordam esse tipo de conteúdo, com a promessa de ensinar técnicas e estudos sobre algoritmos para processos seletivos. Você pode até assistir engenheiros de software arrasando com esse tipo de pergunta nesses canais, na categoria "programação competitiva".

Também há inúmeros sites como **HackerRank**, **LeetCode** e **GeeksforGeeks**, desenvolvidos para ajudar engenheiros a praticar. Na sequência, você pode conferir alguns canais para estudar estrutura de dados e praticar.

- IGotAnOffer: Engineering
  - https://www.youtube.com/@IGotAnOffer-Engineering
- Hello Interview - SWE Interview Preparation
  - https://www.youtube.com/@hello_interview
- Life at Google
  - https://www.youtube.com/@LifeatGoogle
- Clément Mihailescu
  - https://www.youtube.com/@clem
- Sajjaad Khader
  - https://www.youtube.com/@SajjaadKhader
- The Pragmatic Engineer
  - https://www.youtube.com/@pragmaticengineer

Entrevista de System Design

Esse tipo de avaliação normalmente é aplicado para posições a partir de sêniores ou arquitetos de software. Durante essa fase, recrutadores — que normalmente são membros do time — buscam avaliar como o(a) profissional aborda problemas de design de

sistema, tomada de decisões em situações complexas, e como se adapta a mudanças de requisitos. Além disso, analisam como esses profissionais conseguem projetar sistemas que atendem às necessidades do negócio, considerando aspectos como escalabilidade, desempenho, segurança e confiabilidade.

Antes de consumir conteúdos sobre *system design*, procure separar os tópicos que estão relacionados. Por exemplo, alguns tópicos que compõem o *system design* e que você pode estudar de forma separada são:

- Balanceamento de carga;
- Cache (leitura e escrita);
- DNS;
- CDN;
- Bancos de dados: SQL e NoSQL;
- Teorema de CAP;
- Mensageria;
- Blob Store.

Uma boa ideia para treinar esse tipo de situação é consumir conteúdos sobre design de sistema no YouTube e tentar resolver os problemas. Utilize um papel ou ferramentas, como o **Draw.IO**, que possui diversos componentes para desenhar arquitetura de sistemas. A partir disso, apresente a algum colega e tente validar a solução. Colete *feedbacks* e encontre falhas no seu desenho.

Avaliação por projeto prático

Nesse tipo de processo, deve-se criar um projeto a partir de um enunciado que a empresa compartilha, contendo algumas regras de negócios. A empresa concede um prazo, normalmente de uma semana, para a entrega desse projeto, e os candidatos devem submetê-lo no **GitHub** e gerar um *pull request* para que a empresa possa validar.

Esta etapa do processo pode ser eliminatória, dependendo do objetivo de cada empresa. Normalmente, as empresas costumam validar se o projeto atende aos requisitos do enunciado e se foi bem testado. Além disso, é aconselhável que desenvolvedores criem o projeto para executar em um ambiente conteinerizado, o que demonstra conhecimento de boas práticas de desenvolvimento e *deployment*.

Como vimos, criar projetos prepara você para os desafios que a empresa lhe apresentar. Você perderá menos tempo no desenvolvimento, pois já praticou e superou alguns obstáculos anteriormente.

Dicas para se preparar para entrevistas comportamentais

Ao se candidatar a uma vaga, é importante se preparar também em relação à maneira como se comportar durante a conversa e a apresentação do seu currículo. Aqui vão sugestões:

- Pesquise sobre a empresa antes da entrevista para entender melhor seu modelo de negócio, valores e cultura organizacional;
- Tenha exemplos concretos de como suas contribuições geraram resultados positivos em projetos anteriores;
- Seja honesto(a) sobre seus motivos para buscar novas oportunidades, mas mantenha foco nos aspectos positivos de sua experiência anterior;
- Demonstre interesse genuíno fazendo perguntas sobre o negócio, como o fluxo de trabalho funciona e as metodologias utilizadas;
- Esteja preparado para explicar detalhadamente os projetos listados no seu currículo ou no **GitHub**, destacando decisões técnicas, desafios enfrentados e resultados alcançados.

Balanceie competências técnicas e habilidades interpessoais

O mercado de tecnologia valoriza tanto competência técnica quanto habilidades interpessoais. Desenvolva ambos os aspectos de forma

equilibrada, mantendo-se atualizado com tendências tecnológicas e aprimorando sua capacidade de comunicação e colaboração.

Um(a) desenvolvedor(a) completo(a) não é aquele(a) que escreve um excelente código, mas também aquele(a) que consegue trabalhar bem em equipe, comunicar suas ideias claramente e contribuir com um ambiente de trabalho positivo.

Agora que você já conhece as principais técnicas para se destacar em entrevistas, o próximo passo é colocá-las em prática.

Capítulo 17
Equilíbrio sustentável: Como prosperar na carreira tech sem perder a si mesmo

Figura 17.1: Equilíbrio sustentável: Como prosperar na carreira tech sem perder a si mesmo.

Muito provavelmente você já escutou ou leu em algum lugar sobre *burnout*. A síndrome de *burnout*, também conhecida como síndrome do esgotamento profissional, é um distúrbio caracterizado por exaustão, estresse, geralmente resultante de situações desgastantes no trabalho. Essa condição leva à queda na produtividade do indivíduo e afeta sua saúde mental e física.

A OMS (Organização Mundial da Saúde) classifica o *burnout* como um fenômeno ocupacional, resultado de estresse crônico no ambiente de trabalho que não foi adequadamente gerenciado (*World Health Organization*, 2019, traduzido de https://www.who.int/news/item/28-05-2019-burn-out-an-occupational-phenomenon-international-classification-of-diseases). A organização destaca três características principais:

- Sentimentos de esgotamento ou exaustão de energia;

- Aumento da distância mental em relação ao trabalho, ou sentimentos de negativismo ou cinismo relacionados ao trabalho;
- Eficácia profissional reduzida.

Já o site do Ministério da Saúde brasileiro destaca sintomas como:

- Insônia;
- Dificuldades de concentração;
- Sentimentos de incompetência;
- Dor de cabeça frequente;
- Sentimentos de fracasso e insegurança;
- Alteração nos batimentos cardíacos;
- Negatividade constante;
- Sentimentos de derrota e desesperança;
- Dores musculares;
- Pressão alta.

É muito provável que você conheça alguém que já tenha enfrentado algo parecido. Basta conversar com profissionais da área de tecnologia ou acompanhar publicações nas redes sociais sobre o tema para encontrar diversos relatos de experiências reais. Em um artigo publicado pela **ITPro**, especializado em conteúdo sobre o mercado de tecnologia, o alerta é claro:

"O esgotamento profissional de desenvolvedores de software não é um problema novo, mas pouco progresso parece ter sido feito para resolvê-lo nos últimos anos. Quase três quartos (73%) dos desenvolvedores já experimentaram esgotamento profissional em algum momento da carreira". (ITPro, 2024, traduzido de https://www.itpro.com/software/development/burnout-is-now-rife-across-the-software-community-with-almost-half-of-developers-turning-to-self-help-apps)

Afinal, o que leva ao *burnout*? As causas são múltiplas e frequentemente estão interligadas: prazos apertados, cobranças,

pressão do mercado, sobrecarga de demandas, alta rotatividade de empregos, ambientes tóxicos e a insegurança da economia. Tudo isso colabora para o surgimento do *burnout*, mas nem sempre o esgotamento vem só de fatores externos.

Com frequência, somos nós mesmos que construímos esse ambiente interno de pressão e autoexigência extrema, nos cobramos além do que suportamos. A cada erro que cometemos, sentimos culpa e passamos a julgar nossa própria capacidade técnica ou até intelectual. Esse sentimento de inadequação nos leva, por vezes, a tentar "compensar" o erro trabalhando mais, o que alimenta ainda mais o ciclo de esgotamento.

Cobranças internas versus realidade

Apesar de entendermos que errar faz parte do processo de aprendizado, muitas vezes esperamos dominar tudo, como se fôssemos máquinas. Essa cobrança interna pesa na vida dos profissionais. Criamos padrões que são desconexos da realidade do trabalho, do tempo necessário para o aprendizado até a complexidade dos desafios. Somos exigentes quando erramos e, mesmo quando acertamos, buscamos fazer nosso trabalho ainda melhor ou mais rápido.

Ao não aceitar as limitações ou os momentos de oscilação, entramos em um ciclo de autocrítica constante, e pequenos deslizes podem ser vistos como fracassos, em vez de reconhecermos o progresso — o próprio aprendizado que esses erros proporcionam.

Já passei por situações assim, em que não conseguia enxergar os anos de experiência que tenho, as coisas incríveis que já construí e as empresas por onde passei. Só conseguia destacar meus erros e me cobrava ainda mais. Errar era uma frustração que pesava muito.

A melhor forma de encarar tudo isso é olhar para si mesmo e para o que está ao seu redor. Reconhecer seu progresso, sua trajetória e os

obstáculos que já enfrentou. Uma falha não anula seu passado nem sua carreira. Não se cobre tanto.

Cultura do erro

Os erros podem ser valiosos aprendizados, se você souber lidar com eles. Em vez de enxergá-los como algo negativo ou vergonhoso, é importante incorporá-los como parte natural do processo de aprendizagem. Isso é fundamental para o crescimento profissional e pessoal.

Quantas vezes você já deve ter escutado: "Já passei por isso, sei como resolver!"? Essa frase indica que a pessoa enfrentou aquele problema, precisou errar para acertar e transformou a experiência em aprendizado.

Empresas que adotam a cultura do erro criam um ambiente saudável, permitindo que colaboradores não tenham medo de cometer falhas. Isso faz com que pessoas se sintam seguras para tentar coisas novas, sejam mais confiantes sem medo de punições ou cobranças excessivas. Por isso, quando você cometer alguma falha, encare-a como um aprendizado e torne esse processo mais leve.

Vantagens em aceitar ser mediano em vez de excepcional

Demorei um tempo para aceitar que ser mediana na minha carreira não era algo ruim. Sempre acreditei que, para me tornar excepcional, seria preciso estudar muito, e isso é um fato. A genialidade parece ser uma combinação complexa entre predisposição genética, oportunidade e desenvolvimento através de experiências e prática contínua.

Embora seja possível se destacar em uma área específica ou se tornar referência em determinada habilidade, isso geralmente exige um esforço desproporcional para quem não nasceu com essa

propensão favorável. Muitas vezes, o custo desse empenho extremo não compensa os resultados obtidos.

Quero compartilhar aqui que você pode ser bom o suficiente em algo, e isso já é o suficiente para viver bem e trabalhar com qualidade. Você pode escolher ser sua melhor versão todos os dias ao invés de se comparar e tentar se tornar alguém diferente. Isso tornará sua vida mais leve, sem a pressão de buscar o perfeccionismo.

Manter equilíbrio vida pessoal e profissional

Para alcançar resultados, é preciso se esforçar mais, seja estudando ou trabalhando. Há momentos em nossas vidas em que precisamos trabalhar ou estudar mais para atingir determinados objetivos, mas é fundamental manter o equilíbrio para não se sobrecarregar e saber reconhecer os próprios limites.

Antes de mais nada, é importante reconhecer quando estamos ultrapassando nossos limites. Muitas vezes, estamos tão imersos na rotina que não percebemos os sinais que nosso corpo e mente nos enviam.

Alguns indicadores de que talvez seja hora de reavaliar suas prioridades:

- Você sente culpa quando não está sendo produtivo;
- Seus relacionamentos pessoais e sua saúde estão sendo negligenciados;
- Você não consegue encontrar paz ou se concentrar em atividades que gostava.

Estratégias práticas para o equilíbrio

A tecnologia, ao mesmo tempo em que nos conecta, também pode nos aprisionar. É fácil cair nas armadilhas de estar sempre disponível respondendo mensagens ao gerente e resolvendo problemas fora do horário de trabalho, checando e-mails durante as férias ou finais de semana, ou pensando em códigos enquanto deveria estar relaxado. Estabelecer limites é o primeiro passo para equilibrar os dois lados.

E isso não significa ser menos dedicado. É reconhecer que você precisa de um tempo para recarregar suas energias e evitar problemas físicos e mentais. Um desenvolvedor descansado é mais criativo, comete menos erros e toma melhores decisões.

Veja algumas estratégias para manter o equilíbrio:

- Defina horários claros: Estabeleça quando seu dia de trabalho começa e termina. Claro, exceções existem — mas não são a regra;
- Crie rituais de transição: Se você trabalha remotamente de casa, faça rituais que marquem o início e fim do seu dia de trabalho. Respeite os horários das refeições e pausas, pratique exercícios regularmente antes ou depois do horário de trabalho;
- Pratique o "não": Nem tudo precisa ser aceito de imediato — como tarefas, projetos extras ou cursos adicionais. Avalie e pense "eu consigo encaixar isso na minha rotina?" ou "eu preciso fazer isso agora?". Saber dizer "não" é estabelecer limites. Se você aceitar tudo, tudo se torna prioridade pra você;
- Cultive um hobby: Tenha lazer, alguma atividade que tire você da frente do computador ou do celular. Experiências novas podem até mesmo influenciar positivamente seu trabalho, trazendo perspectivas únicas transferíveis.

Encontrar equilíbrio não significa dividir seu tempo igualmente entre trabalho e vida pessoal todos os dias. Significa ter a flexibilidade para se dedicar mais intensamente ao trabalho quando necessário,

sabendo que você também terá momentos para cuidar de outros aspectos da sua vida.

Sua jornada como desenvolvedor(a) continua

Ao longo deste livro, exploramos juntos, de diversas maneiras, a carreira em desenvolvimento de software. Abordamos desde os primeiros passos — conhecendo as principais funções envolvidas — até questões mais complexas, como o equilíbrio entre vida pessoal e profissional. Cada capítulo foi pensado para acompanhar você em diferentes momentos da sua trajetória.

Muitos dos assuntos abordados neste livro são dúvidas frequentes que escuto de profissionais da área de tecnologia — desde desabafos até discussões técnicas relevantes compartilhadas entre amigos. O cenário e os negócios podem mudar, mas as experiências sempre têm algo a acrescentar à nossa própria trajetória.

Vimos que ser um(a) ótimo(a) desenvolvedor(a) vai muito além de dominar linguagens de programação ou frameworks. É sobre desenvolver uma mentalidade de crescimento contínuo, saber comunicar suas ideias, atuar bem em equipe e, principalmente, ter uma visão macro do negócio e arquitetura do sistema. Com o avanço da Inteligência Artificial que nos ajuda com a criação e correção de código, nosso trabalho se direciona para o conhecimento do negócio e o desenho de arquitetura.

A tecnologia evolui rapidamente. Novas ferramentas, metodologias e paradigmas surgem constantemente. Entretanto, os fundamentos que discutimos, bem como a organização pessoal e o desenvolvimento de *soft skills* são pilares que permanecerão relevantes ao longo da sua carreira, e IA alguma substituirá isso.

Lembre-se de que não existe fórmula mágica ou curso que tornará você um(a) desenvolvedor(a) de sucesso em pouco tempo. Cada desenvolvedor(a) tem sua própria jornada, seus próprios desafios e

suas conquistas. O importante é ser resiliente e estar aberto ao aprendizado.

Continue se atualizando, aprendendo e se conectando com a comunidade de tecnologia. O mercado precisa de desenvolvedores como você — que não apenas escrevem código elegante e funcional, mas também entregam valor ao negócio, acompanham o andamento da arquitetura, monitoram a aplicação em ambientes produtivos e buscam corrigir gargalos que possam surgir.

Boa sorte na sua jornada!

Capítulo 18
Referências bibliográficas

ATLASSIAN. Controle de versão Git. Disponível em: https://www.atlassian.com/br/git/tutorials/what-is-git#:~:text=De%20longe%2C%20o%20sistema%20de,kernel%20do%20sistema%20operacional%20Linux. Acesso em: 6 jul. 2025.

DUHIGG, Charles. *O poder do hábito*. Tradução de Rafael Mantovani. 1. ed. Rio de Janeiro: Objetiva, 2012.

EXAME. Investimento das Big Techs em IA 2025. Disponível em: https://exame.com/inteligencia-artificial/big-techs-devem-investir-us-320-bilhoes-em-ia-em-2025/?utm_source=copiaecola&utm_medium=compartilhamento. Acesso em: 6 jul. 2025.

EXTREME PROGRAMMING. A gentle Introduction. Disponível em: http://www.extremeprogramming.org/. Acesso em: 6 jul. 2025.

FACEBOOK ENGINEERING. Meta cultura code review. 2022. Disponível em:

https://engineering.fb.com/2022/11/16/culture/meta-code-review-time-improving. Acesso em: 6 jul. 2025.

FORBES. O Que é o DeepSeek e Como Ele Afeta a Relação entre China e EUA? 2025. Disponível em: https://forbes.com.br/forbes-tech/2025/01/o-que-e-o-deepseek-e-como-ele-afeta-a-relacao-entre-china-e-eua/. Acesso em: 6 jul. 2025.

FORTUNE BUSINESS INSIGHTS. MLOps Market. Disponível em: https://www.fortunebusinessinsights.com/pt/mlops-market-108986. Acesso em: 6 jul. 2025.

GITHUB. Innovation Graph. Disponível em: https://innovationgraph.github.com/. Acesso em: 6 jul. 2025.

GOOGLE. Guia Google boas práticas code review. Disponível em: https://google.github.io/eng-practices/review/reviewer/speed.html. Acesso em: 6 jul. 2025.

GITHUB. Pesquisa sobre o impacto positivo na produtividade dos desenvolvedores. Disponível em: https://github.blog/news-insights/research/research-

quantifying-github-copilots-impact-on-developer-productivity-and-happiness/. Acesso em: 6 jul. 2025.

HUNT, Andrew; THOMAS, David. *O programador pragmático*: de aprendiz a mestre. Tradução de Aldir José Coelho Corrêa da Silva. Revisão técnica de Henrique J. Brodbeck. 1. ed. São Paulo: Bookman, 2010.

MARIOTTO, Gladys. *Já entendi*. 1. ed. São Paulo: Planeta do Brasil, 2015.

NETFLIX TECHNOLOGY BLOG. Full Cycle Developers at Netflix — Operate What You Build. 2018. Disponível em: https://netflixtechblog.com/full-cycle-developers-at-netflix-a08c31f83249. Acesso em: 6 jul. 2025.

NINLABS RESEARCH. Programmer Interrupted. 2013. Disponível em: http://blog.ninlabs.com/2013/01/programmer-interrupted/. Acesso em: 6 jul. 2025.

RIC.CPS.SP.GOV.BR. O Inglês como idioma da área de tecnologia da Informação. Disponível em: https://ric.cps.sp.gov.br/handle/123456789/21438. Acesso em: 6 jul. 2025.

RUBBER DUCK DEBUGGING. Disponível em: https://rubberduckdebugging.com/. Acesso em: 6 jul. 2025.

SCRUMGUIDES. Scrum Guide. 2013. Disponível em: https://scrumguides.org/. Acesso em: 6 jul. 2025.

SOFTWARE TESTING HELP. Software development life cycle. Disponível em: https://www.softwaretestinghelp.com/Software-development-life-cycle-sdlc/. Acesso em: 6 jul. 2025.

SOFTWARE.COM. Code time report. Disponível em: https://www.software.com/reports/code-time-report. Acesso em: 6 jul. 2025.

SPICEWORKS. The 2025 State of IT. Disponível em: https://www.spiceworks.com/research/it-report/. Acesso em: 6 jul. 2025.

STACKOVERFLOW. Five years ago, StackOverflow launched then, a miracle occured. Stack Overflow Blog, 16 set. 2013. Disponível em: https://stackoverflow.blog/2013/09/16/five-years-ago-stack-overflow-launched-then-a-miracle-occurred/. Acesso em: 6 jul. 2025.

STACKOVERFLOW. The hardest part of building Software is not coding. Stack Overflow Blog, 26 jun. 2023. Disponível em: https://stackoverflow.blog/2023/06/26/the-hardest-part-of-building-Software-is-not-coding-its-requirements/. Acesso em: 6 jul. 2025.

THE NEW STACK. The anatomy of slow code reviews. Disponível em: https://thenewstack.io/the-anatomy-of-slow-code-reviews/. Acesso em: 6 jul. 2025.

VADIM KRAVCENKO. Things They didn't teach you. Disponível em: https://vadimkravcenko.com/shorts/things-they-didnt-teach-you/. Acesso em: 6 jul. 2025.